Kamerabasert navigasjon ved hjelp av landemerker med ukjent avstand fra kameraplanet

Masteroppgave

Simen Tvedt Engen

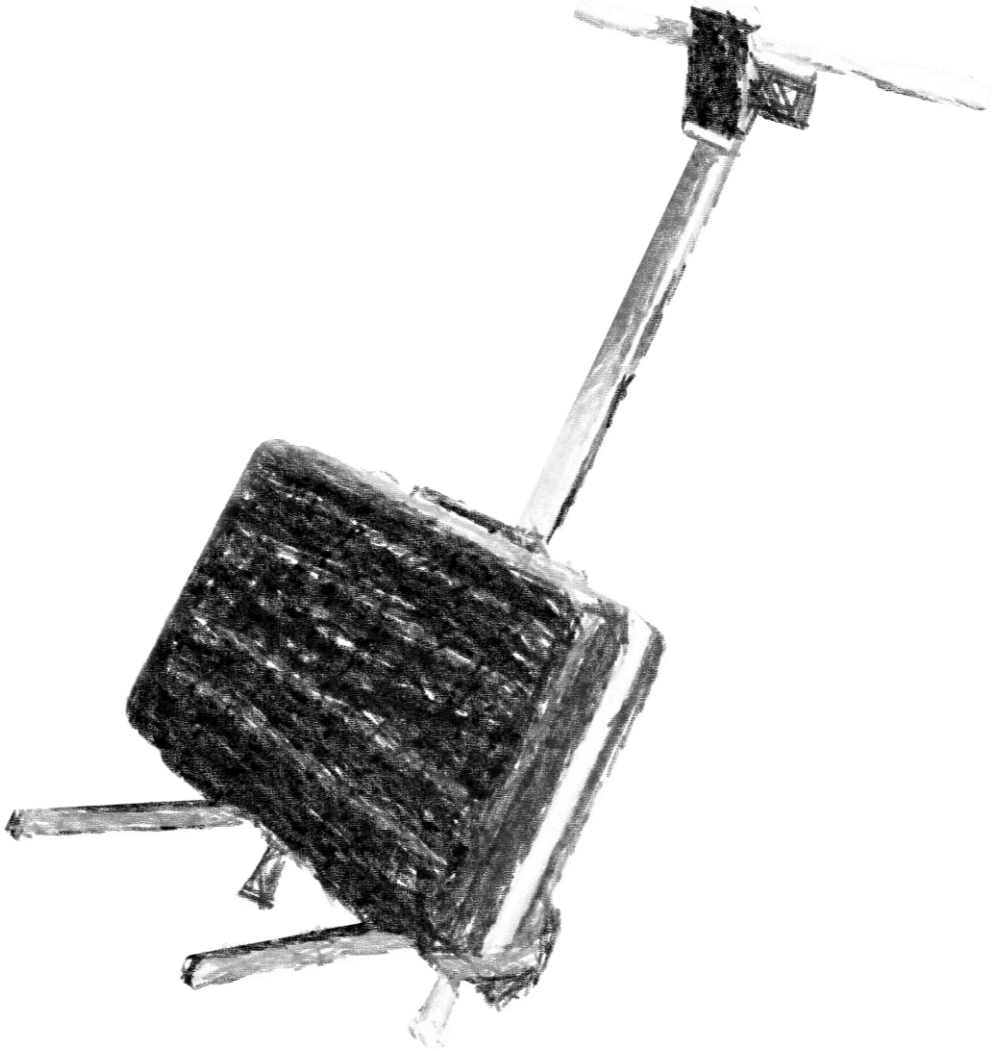
Masteroppgave Fysisk Institutt

UNIVERSITETET I OSLO

27.05.2015

Kamerabasert navigasjon ved hjelp av landemerker med ukjent avstand til kameraplanet

Vil kamerabase navigasjon gi et bedre posisjon- og orienteringsestimat ved å implementere treghetsmålinger fra en lavkost treghetsplattform?



2015

Kamerabasert navigasjon ved hjelp av landemerker med ukjent avstand til kameraplanet

http://www.duo.uio.no/

Trykk: Forsvarets Forskningsinstitutt - Trykkeriet

FORORD

Denne oppgaven er skrevet som det avsluttende arbeidet i min mastergrad i Kybernetikk under studieprogrammet Elektronikk og datateknologi ved Fysisk institutt, Matematisk Naturvitenskaplig fakultet, Universitetet i Oslo.

Jeg vil rette en stor takk til mine veiledere Oddvar Hallingstad ved UNIK (Universitetssenteret på Kjeller) og Kjetil Bergh Ånonsen ved FFI (Forsvarets Forskningsinstitutt) som har gitt meg svært god veiledning under min periode som masterstudent og under arbeidet med min masteroppgave.

Følgende dataverktøy er brukt i prosjektet:

- Microsoft Word
- Microsoft Excel
- Microsoft Visio
- Microsoft Project
- Microsoft Mathematics
- Microsoft Snipping Tool
- MATLAB
- VLC Media Player
- Multi Commander
- IrfanView
- Sensor Kinetics Pro (Android applikasjon)

Kjeller, Mai 2015

_____

Simen Tvedt Engen

SAMMENDRAG

Kamerabasert navigasjon er en metode for å estimere posisjon og orientering til en plattform ved hjelp av informasjon fra optiske kamerabilder. Det kan være aktuelt i flere situasjoner. Eksempler på dette kan være der det ikke finnes eller er dårlig GPS dekning som innendørs, i et urbant miljø, under vann eller i situasjoner der hvor GPS signalet blir "jammet".

Oppgavens mål er å gjøre seg kjent med kamerabasert navigasjon og videre se om implementering av akselerasjon- og vinkelhastighetsmålinger fra lavkost akselerometer og gyroskop vil forbedre navigasjonsresultatet.

Oppgaven tar for seg et eksisterende kamerabasert navigasjonsprogram som er programmert i MATLAB. Programmet blir utvidet for å implementere målinger fra en treghetsnavigasjonsplattform. Det kamerabaserte navigasjonsprogrammet bruker et ordinært digitalt monokamera og estimerer posisjonen i skala (når et minimum antall bildemerker blir brukt). Programmet bruker en bildebehandlingsalgoritme for å finne karakteristiske bildemerker i bildescenen. Ved å finne de samme bildemerkene i neste bilde og se på forflytningen, blir bevegelsen estimert på bakgrunn av dette.

Oppgaven setter først opp det eksisterende kameranavigasjonsprogrammet til å fungere med egne bilder. Deretter blir programmet utvidet slik at lineærakselerasjon og vinkelhastighet blir implementert. Akselerasjon og vinkelhastighet blir målt ved hjelp av et akselerometer og et gyroskop på en mobiltelefon.

Resultatet viser at ved å utvide kameranavigasjonsprogrammet kan posisjonen og orienteringen estimeres med en bedre nøyaktighet, samtidig som antall bildemerker kan reduseres. Dette fører til at algoritmen reduserer behovet av datakraft. Men det viser seg at ved et lite feilestimat av orienteringen skaper den målte akselerasjonen stor feil i posisjonsestimatet. Dette er på grunn av feilestimatet gjør at målt akselerasjon blir rotert feil og vil gi akselerasjon i feil retning. Rapporten foreslår at et 3-akset magnetometer kan bli brukt for å utbedre problemet.

INNHOLDSFORTEGNELSE

NOMENKLATURLISTE

DOF	Degrees of freedom
FFI	Forsvarets forskningsinstitutt
FPS	Frames per second “bilderuter per sekund”
GNU	General Public License
PGM	Portable Graymap Format
UKF	Utvidet Kalman Filter/Utvidet Kalman Filter
UNIK	Universitetssenteret på Kjeller
RANSAC	Random sample consensus
SLAM	Simultaneous Localization and Mapping
SNR	Signal-to-Noise ratio

FIGURER

TABELLER

1 INNLEDNING

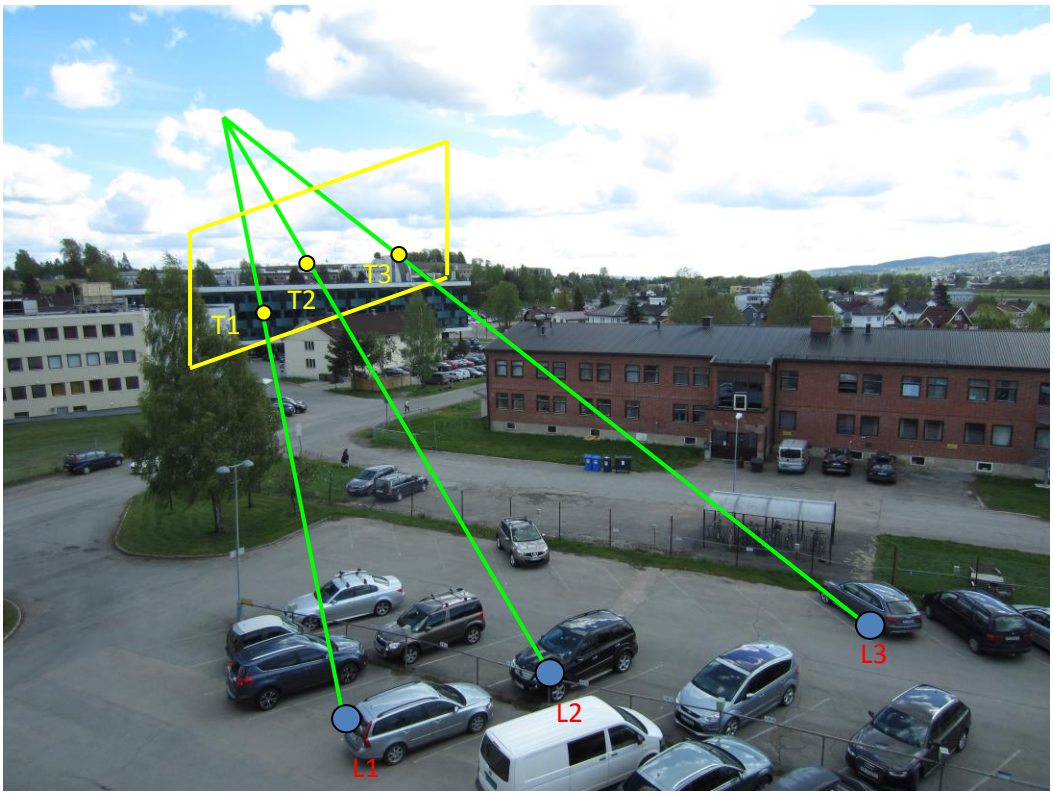
Kamerabasert navigasjon er en metode for å estimere posisjon og orientering til en plattform ved hjelp av informasjon fra optiske kamerabilder. Dette kan enten gjøres som en frittstående navigasjonsmetode, eller i kombinasjon med et annet navigasjonssystem, for eksempel et treghetsnavigasjonssystem. I denne masteroppgaven er det brukt en eksisterende kamerabasert navigasjonsalgoritme som bruker bilder fra et monokamera. Metoden går ut på å følge bevegelsen til karakteristiske bildemerker (se Figur 1-1) i en sekvens av kamerabilder og estimere bevegelsen på bakgrunn av dette. Oppgaven konsentrerer seg om det tilfellet der man ikke kjenner avstanden fra kameraet til landmerkene (se Figur 1-1), og ingen objekter med kjent størrelse i bildescenen. Algoritmen som er sett på i oppgaven tar for seg såkalt invers dybde-parametrisering som er en mulig løsning på dette problemet.

Det er flere fordeler ved å bruke kamerabasert navigasjon. Det kan være i situasjoner der det ikke finnes eller er svært dårlig GPS dekning som for eksempel innendørs, i et urbant miljø, under vann eller i situasjoner der hvor GPS signalet blir jammet. Kamerabasert navigasjon kan være støttende til GPS navigasjon eller være hovednavigasjonskilden. Det er også en mulighet å bruke treghetsnavigasjon alene, men rimelige treghetsnavigasjonsplattformer har en uakseptabelt høy drift etter kort tid.

En annen fordel med kamerabasert navigasjon er at digitale kameraer er relativt billig sammenlignet med nøyaktige treghetsnavigasjonsplattformer. En kombinasjon av kamerabasert navigasjon og en rimelig treghetsnavigasjonsplattform ville blitt et billig navigasjonsalternativ. Det er derfor ønskelig å se på kamerabasert navigasjon alene, og deretter se på kamerabasert navigasjon med implementert målinger fra en lavkost treghetsnavigasjonsplattform for å se om det vil medføre en forbedring i posisjonsestimatet.

Denne masteroppgaven tar for seg nettopp det å bruke et lavkost digitalkamera og treghetsmålinger fra akselerometer og gyroskop i en mobiltelefon. Oppgaven bruker et eksisterende kamerabasert navigasjonsprogram som er implementert i MATLAB. Programmet løser SLAM-problemet (Simultaneous Localization and Mapping), som vil si at det lokaliserer sin egen posisjon samtidig som den bygger opp et kart av omgivelsene. Programmet er utviklet og implementert av [1], og er gitt ut under GNU og kan finnes i [2].

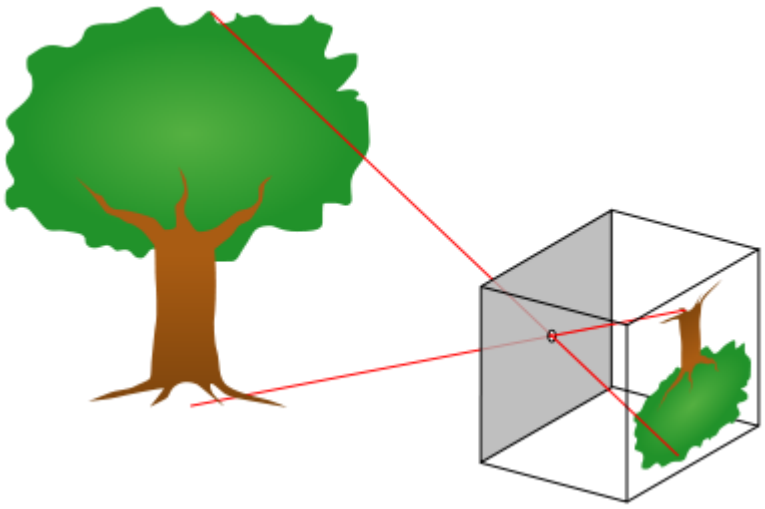
Kapittel 2 utleder en matematisk modell av et pinhole-kamera, som er en modell av et kamera i sin enkleste from uten linse. I kapittel 3 presenteres et matematisk grunnlag for rapporten. Det eksisterende kamerabaserte navigasjonsprogrammet og utvidelsen av programmet er presentert i kapittel 4. Kapittel 5 inneholder en beskrivelse av testoppsettet og forberedelser for å utføre eksperimenter med kamerabasert navigasjon med og uten utvidelsen. I kapittel 6 presenteres resultatene fra eksperimentene, før de blir diskutert i kapittel 7. Rapporten avsluttes med kapittel 8 og 9 som inneholder konklusjon og forslag til videre arbeid.



Figur 1-1 Landemerke L er et punkt på en overflate i scenen som antas å ligge i ro. Bildemerke T er sentralprojeksjonen av et landemerke.

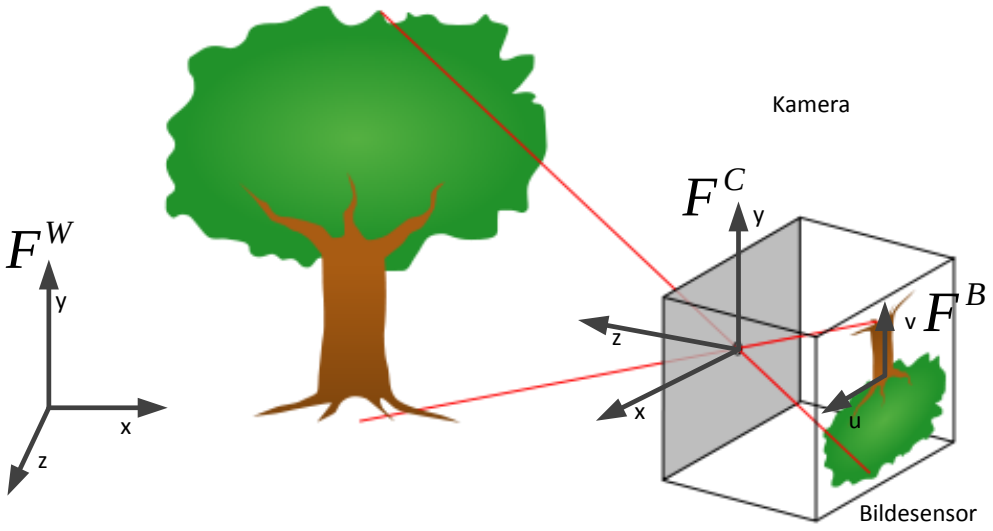
2 PINHOLE-KAMERAMODELL

For at et kamera skal kunne brukes i navigasjon, er det nødvendig med en matematisk modell av kameraet. En eksakt modell av et ordinært kamera er matematisk komplisert på grunn av forvrenging i linen. For å forenkle modellen blir det derfor isteden ofte brukt en pinhole-kameramodell [3]. Et pinhole-kamera (vist i Figur 2-1) er et kamera i sin enkleste form uten linse. Forenklingene som gjøres kompenseres ved å bruke en koordinattransformasjon på bildekoordinatene i etterkant. Underkapitlene presenterer utledingen av den matematiske modellen for et pinhole-kamera.



Figur 2-1 Prinsippet til et pinhole-kamera [4]

For å modellere et pinhole-kamera er det nødvendig å operere med flere rammer. Figur 2-2 viser de ulike rammene. F^W er verdensrammen, F^C er kameraets ramme og F^B er bildesensorrammen.



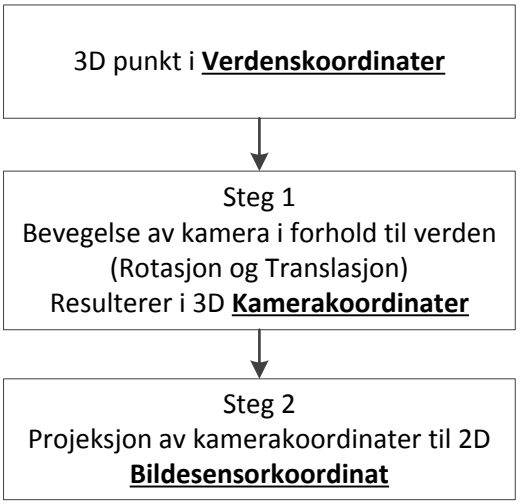
Figur 2-2 Rammene i kameramodellen

En forklaring av rammene vist i Figur 2-2 er presentert i Tabell 2-1.

Tabell 2-1 Koordinatsystemene i kameramodellen

Ramme	**Parameter**	**Navn**	**Enhet**
F^W	(X_W, Y_W, Z_W)	3D verdenskoordinat	$[m]$
F^C	(X_C, Y_C, Z_C)	3D kamerakoordinat	$[m]$
F^B	(u, v)	2D bildesensorkoordinat	[piksel]
-	Z_C	Hovedakse - er den vektor vinkelrett på bildet, og skjærer med kamerasenteret	$[m]$

Bildedannelsen i et pinhole-kamera skjer i rekkefølgen som er vist i Figur 2-3.



Figur 2-3 Bildedannelsesprosessen i et pinhole-kamera

2.1 Indre parametere i kameramodellen

De indre parameterne som inngår i kameramodellen er de parameterne som er innvendig i kameraet. Disse parameterne er tidsinvariante.

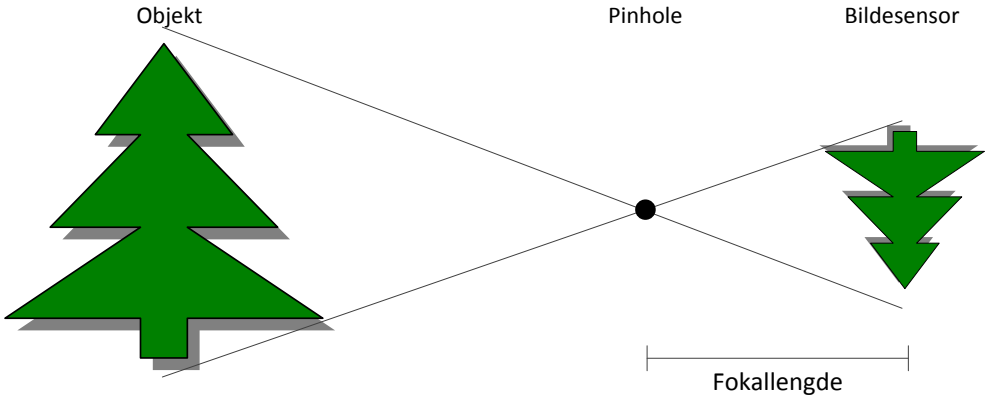
2.1.1 De indre parameterne

De indre parameterne er forklart i Tabell 2-2 vist i Figur 2-4 og Figur 2-5.

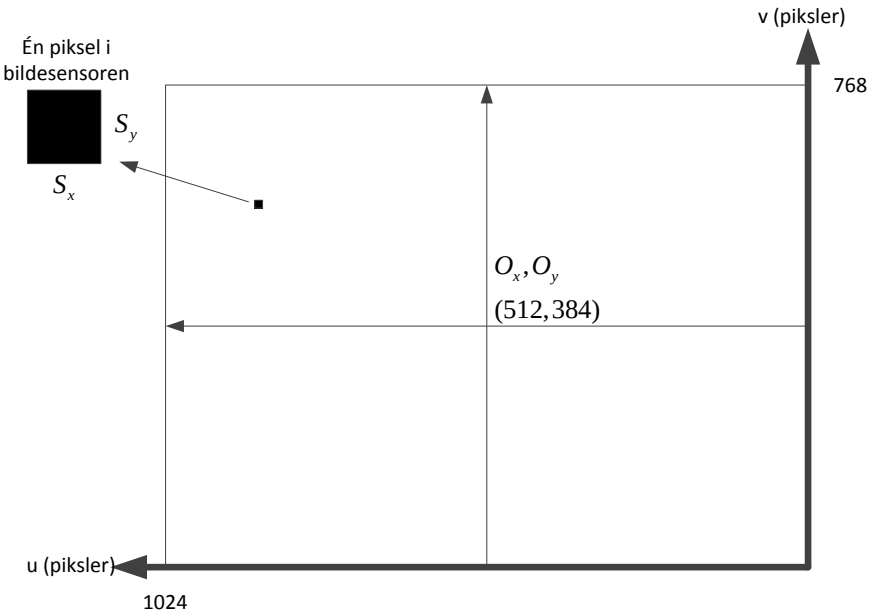
Tabell 2-2 Forklaring av de indre parameterne i et pinhole-kamera.

Parameter	**Navn**	**Enhet**	**Beskrivelse**
f	Fokallengde (også kalt Brennvidde)	$[mm]$	Avstanden fra planet der et bilde dannes til det optiske sentrum i en enkel linse eller objektivkonstruksjon når denne er innstilt på uendelig avstand (∞)[5].

S_x, S_y	Piksel-størrelsen	$[\mu m]$	Høyde og bredde på én piksel på bildesensoren, S_x, S_y er ofte like. Figur 2-5 viser en illustrasjon av én piksel.
O_x, O_y	Bildesenteret	$[u, v]$	Midtpunktet til bildesensoren. Eksempel: hvis en bildesensor på 1024x768 piksler montert eksakt i senter av det projiserte bilde, vil O_x, O_y bli $[1024/2 \quad 768/2] = [512 \quad 384]$ i piksler. Figur 2-5 viser en illustrasjon av bildesenteret.



Figur 2-4 Fokallengden i et pinhole-kamera



Figur 2-5 Viser en bildesensor med to av de statiske parameterne. I dette tilfellet er bildesensoren montert eksakt i senter av det projiserte bilde og vil derfor bli (512,384).

2.1.2 Projeksjon av 3D kamera koordinater til bildeplanet

Ut fra de indre parameterne kan 3D koordinatene oppgitt i F^C projiseres til 2D koordinater oppgitt i F^B. Projisering fra F^C til F^B er illustrert i Figur 2-6 og kan regnes ut slik:

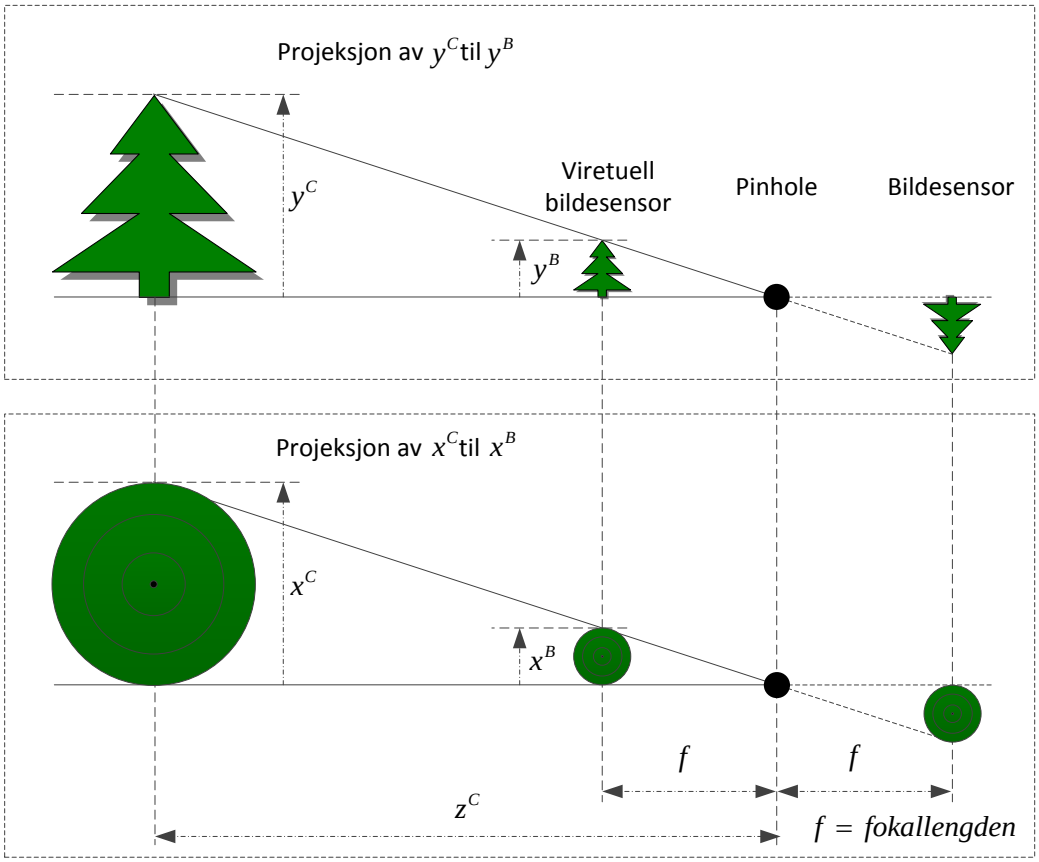
$$x^B = f\frac{x^C}{z^C} \tag{2.1}$$

$$y^B = f\frac{y^C}{z^C} \tag{2.2}$$

Ved bruk av størrelsen på et piksel S_x, S_y og bildesenteret O_x, O_y kan x^B og y^B omregnes til u og v som er F^B oppgitt i pikselkoordinatverdier:

$$u = \frac{f}{S_x}\frac{x^C}{z^C} + o_x \tag{2.3}$$

$$v = \frac{f}{S_y}\frac{y^C}{z^C} + o_y \tag{2.4}$$



Figur 2-6 Projeksjon av 3D kamera koordinater (X, Y, Z) til bildeplanet (u, v). Nedre del av bilde viser treet sett fra fugleperspektiv.

2.1.3 Indre parameter-matrise $\mathbf{M}_{indre}$

Ligning (2.3) og (2.4) er ulineære på grunn av $\frac{1}{z_C}$. Men ligningene kan omskrives til en lineær matrise $\mathbf{M}_{\text{int}}$:

$$\begin{bmatrix} su \\ sv \\ s \end{bmatrix} = \begin{bmatrix} f/s_x & 0 & o_x \\ 0 & f/s_y & o_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_C \\ y_C \\ z_C \end{bmatrix} = \mathbf{M}_{indre} \begin{bmatrix} x_C \\ y_C \\ z_C \end{bmatrix} \tag{2.5}$$

Hvor s er en hjelpevariabel. For å finne u og v må man løse:

$$u = \frac{su}{s} \tag{2.6}$$

$$v = \frac{sv}{s} \tag{2.7}$$

Ved å sette:

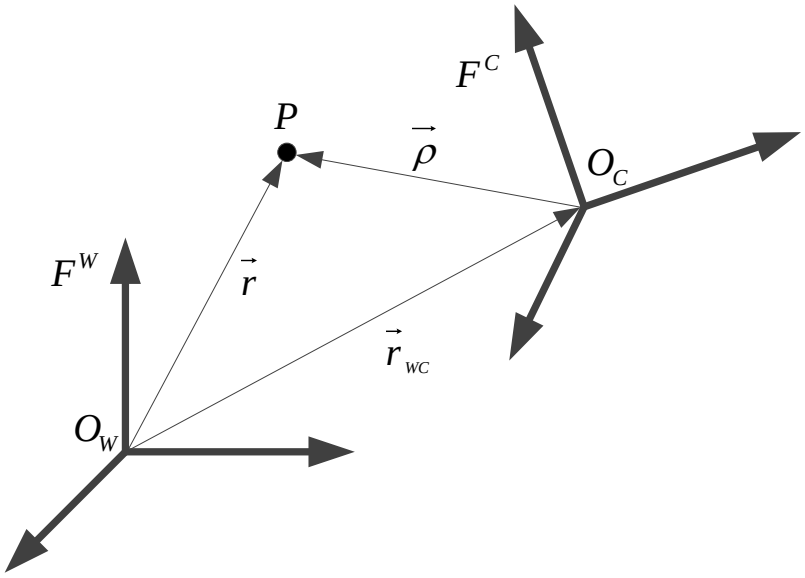
$$f_{piks} = \frac{f}{s_x} = \frac{f}{s_y} \tag{2.8}$$

som er fokallengden oppgitt i piksler, Kan $\mathbf{M}_{indre}$ skrives:

$$\mathbf{M}_{indre} = \begin{bmatrix} f_{piks} & 0 & o_x \\ 0 & f_{piks} & o_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.9}$$

2.2 Ytre parametere i kameramodellen

Rotasjon og translasjon er de ytre eller de dynamiske parameterne i en kameramodell. Dvs. kameraet kan rotere og flytte på seg i rommet. Figur 2-7 viser systemet med F^W og F^C som er henholdsvis verdensrammen og kamerarammen.



Figur 2-7 Verdensrammen og kameramodellen med de aktuelle vektorene.

Hensikten med kameramodellen er å finne posisjonen til punktet P sett fra F^C. Punket P har kun kjent posisjon sett fra F^W. Det er derfor nødvendig å omregne posisjonen fra F^W til F^C. Figur 2-7 og likning (2.10) viser at punktet P kan representeres på to ulike måter:

$$P = O_W + \vec{r} = O_C + \vec{\rho} \tag{2.10}$$

Ved deretter å flytte origoene og vektorene på hver side:

$$O_W - O_C = \vec{\rho} - \vec{r} \tag{2.11}$$

og videre multiplisere med -1 på hver side:

$$O_C - O_W = \vec{r} - \vec{\rho} = \vec{r}_{WC} \tag{2.12}$$

Dette er vektoren mellom de to rammene. Siden det er ønskelig å finne posisjonen til punktet P sett fra F^C løses likning (2.12) med hensyn på $\vec{\rho}$:

$$\vec{\rho} = \vec{r} - \vec{r}_{WC} \tag{2.13}$$

$\vec{\rho}$ representert i F^C blir dermed:

$$\boldsymbol{\rho}^C = \mathbf{r}^C - \mathbf{r}_{WC}^C \tag{2.14}$$

Siden vektorene $\vec{r}$ og $\vec{r}_{WC}$ er kun kjent i F^W må en rotasonsmatrise brukes for å representere disse i F^C:

$$\boldsymbol{\rho}^C = \mathbf{R}_W^C(\mathbf{r}^W - \mathbf{r}_{WC}^W) \tag{2.15}$$

Hvor:

- $\mathbf{r}^W$ er en kolonnematrise som representerer posisjonen til punkt P oppgitt i verdenskoordinater

- $\boldsymbol{\rho}^C$ er en kolonnematrise som representerer posisjonen til punkt P oppgitt i kamerakoordinater
- $\mathbf{r}_{WC}^W$ er en kolonnematrise som representerer translasjonen av kamerarammen fra verdensrammen
- $\mathbf{R}_W^C$ er en $n \times n$ rotasjonsmatrise som roterer vektorer fra verdensrammen til kamerarammen

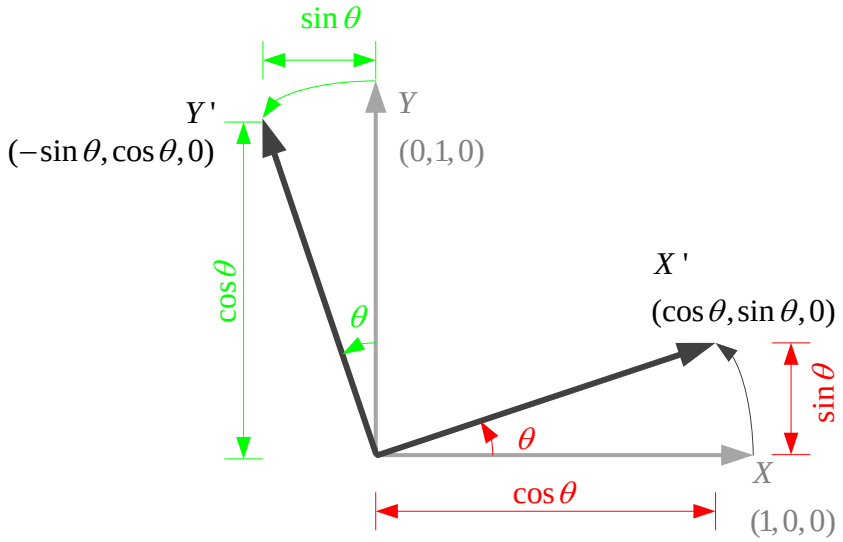
$$\mathbf{R}_W^C = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{2.16}$$

Likning (2.15) er likningen for den ytre modelleringen av kameraet.

Underkapitlene beskriver utledningen av disse matrisene. Rotasjon- og translasjonsmatrisen settes til slutt sammen til en transformasjonsmatrise som er en homogen representasjon av punkters koordinat.

2.2.1 Rotasjonsmatrise

Rotasjonen er bygget opp ved først å se på rotasjonen rundt en og en akse, og deretter sette dem sammen til en rotasonsmatrise. Figur 2-8 viser positiv rotasjon (høyrehåndsregelen) θ_Z rundt Z-aksen mens X- og Y-aksen har ingen rotasjon (θ_X og $\theta_Y = 0$).



Figur 2-8 Viser positivrotasjon rund Z-aksen

Ved en positivrotasjon rundt Z-aksen med θ_Z grader, vil den nye X-aksen gå fra (1,0,0) til (cos(θ_Z), sin(θ_Z), 0), og Y-aksen gå fra (0,1,0) til (-sin(θ_Z), cos(θ_Z), 0). Ved å samle dette til en rotasjonsmatrise for Z-aksen fra F^C til F^W blir det:

$$\mathbf{R}_C^W(\theta_z) = \begin{bmatrix} \cos\theta_z & -\sin\theta_z & 0 \\ \sin\theta_z & \cos\theta_z & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.17}$$

Ligning (2.15) viser at det er nødvendig med en rotasjonsmatrise fra F^W til F^C. Ved å vise at:

$$\mathbf{R}_C^W(\theta_z)\mathbf{R}_C^{W-1}(\theta_z) = I \tag{2.18}$$

betyr det at $\mathbf{R}_W^C(\theta_z)$ blir:

$$\mathbf{R}_W^C(\theta_z) = \mathbf{R}_C^{W-1}(\theta_z) = \begin{bmatrix} \cos\theta_z & \sin\theta_z & 0 \\ -\sin\theta_z & \cos\theta_z & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.19}$$

Med samme fremgangsmåte kan rotasjonsmatrisene rundt X- og Y-aksen også regnes ut. Resultatet blir:

$$\mathbf{R}_W^C(\theta_y) = \begin{bmatrix} \cos\theta_y & 0 & -\sin\theta_y \\ 0 & 1 & 0 \\ \sin\theta_y & 0 & \cos\theta_y \end{bmatrix} \tag{2.20}$$

$$\mathbf{R}_W^C(\theta_x) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_x & \sin\theta_x \\ 0 & \sin\theta_x & \cos\theta_x \end{bmatrix} \tag{2.21}$$

Disse tre rotasjonsmatrisen kan settes sammen til én rotasjonsmatrise:

$$\mathbf{R}_W^C = \mathbf{R}_W^C(\theta_x, \theta_y, \theta_z) \tag{2.22}$$

$$\mathbf{R}_W^C = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_x & \sin\theta_x \\ 0 & \sin\theta_x & \cos\theta_x \end{bmatrix} \begin{bmatrix} \cos\theta_y & 0 & -\sin\theta_y \\ 0 & 1 & 0 \\ \sin\theta_y & 0 & \cos\theta_y \end{bmatrix} \begin{bmatrix} \cos\theta_z & \sin\theta_z & 0 \\ -\sin\theta_z & \cos\theta_z & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.23}$$

$$\mathbf{R}_W^C = \begin{bmatrix} \mathrm{c}\theta_y c\theta_z & c\theta_y s\theta_z & -s\theta_y \\ c\theta_z s\theta_x s\theta_y - c\theta_x s\theta_z & c\theta_x c\theta_z + \mathrm{s}\theta_x s\theta_y s\theta_z & c\theta_y s\theta_x \\ s\theta_x s\theta_z + c\theta_x c\theta_z s\theta_y & c\theta_x s\theta_y s\theta_z - c\theta_z s\theta_x & c\theta_x c\theta_y \end{bmatrix} \tag{2.24}$$

Denne matrisen vil gjøre om verdenskoordinater til kamera koordinater:

$$\begin{bmatrix} X_C \\ Y_C \\ Z_C \end{bmatrix} = \mathbf{R}_W^C \begin{bmatrix} X_W \\ Y_W \\ Z_W \end{bmatrix} \tag{2.25}$$

2.2.2 Ytre parameter-matrise $\mathbf{M}_{ytre}$

Likning (2.15) er en inhomogen likning av det dynamiske systemet beregner 3D verdens koordinater til 3D kamera koordinater. Transformasjonsmatrisen er en homogen representasjon av denne likningen og utfører dermed en rotasjon og en translasjon.

Likning (2.15) skrevet ut gir:

$$\begin{bmatrix} x^C \\ y^C \\ z^C \end{bmatrix} = \left[\mathbf{R}_{W\,3\times3}^C \left[\begin{bmatrix} x^W \\ y^W \\ z^W \end{bmatrix} - \mathbf{r}_{WC}^W \right] \right] \tag{2.26}$$

Hvor:

$$\mathbf{R}_W^C = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \text{ og } \mathbf{r}_{WC}^W = \begin{bmatrix} x^W \\ y^W \\ z^W \end{bmatrix} \tag{2.27}$$

Likning (2.26) kan skrives som:

$$\begin{bmatrix} x^C \\ y^C \\ z^C \end{bmatrix} = \mathbf{R}_{W\ 3\times3}^C \begin{bmatrix} x^W \\ y^W \\ z^W \end{bmatrix} - \left[\mathbf{R}_{W\ 3\times3}^C \mathbf{r}_{WC}^W \right] \tag{2.28}$$

For å gjøre (2.28) til en homogen likning kan matrisene utvides med en rad med 1 i bunn på begge sider:

$$\begin{bmatrix} x^C \\ y^C \\ z^C \\ 1 \end{bmatrix}_{4\times1} = \left[\mathbf{R}_{W\ 3\times3}^C \quad -\left(\mathbf{R}_{W\ 3\times3}^C \mathbf{r}_{WC}^W \right)_{3\times1} \right] \begin{bmatrix} x^W \\ y^W \\ z^W \\ 1 \end{bmatrix}_{4\times1} = \mathbf{T}_{W\ 3\times4}^C \begin{bmatrix} x^W \\ y^W \\ z^W \\ 1 \end{bmatrix}_{4\times1} = \mathbf{M}_{ytre} \begin{bmatrix} x^W \\ y^W \\ z^W \\ 1 \end{bmatrix}_{4\times1} \tag{2.29}$$

Likning (2.29) er den homogene representasjonen av den inhomogene likningen (2.15). $\mathbf{T}_W^C$ er transformasjonsmatrisen som kalles $\mathbf{M}_{ytre}$.

2.3 Sette sammen indre parameter-matrise $\mathbf{M}_{indre}$ og ytre parameter-matrise $\mathbf{M}_{ytre}$ til $\mathbf{P}$

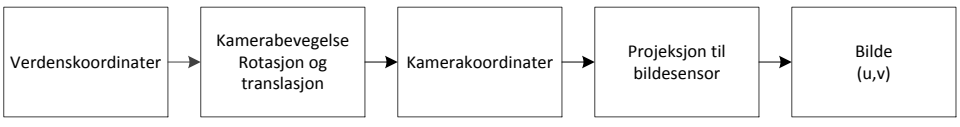
Kameraets indre og ytre modell kan settes sammen til kameraets modell $\mathbf{P}$:

$$\begin{bmatrix} su \\ sv \\ s \end{bmatrix} = \mathbf{M}_{indre} \mathbf{M}_{ytre} \begin{bmatrix} x^W \\ y^W \\ z^W \\ 1 \end{bmatrix} \tag{2.30}$$

$$\begin{bmatrix} su \\ sv \\ s \end{bmatrix} = \mathbf{P}_{3\times4} \begin{bmatrix} x^W \\ y^W \\ z^W \\ 1 \end{bmatrix} \tag{2.31}$$

Kameramodellen $\mathbf{P}$ vil transformere 3D-verdenskoordinater over til det 2D koordinat-systemet til kameraets bildeplan.

Veien fra verdenskoordinater til kamerakoordinater er oppsummert i Figur 2-9 og modellen oppsummeres i likning (2.30) og (2.31).



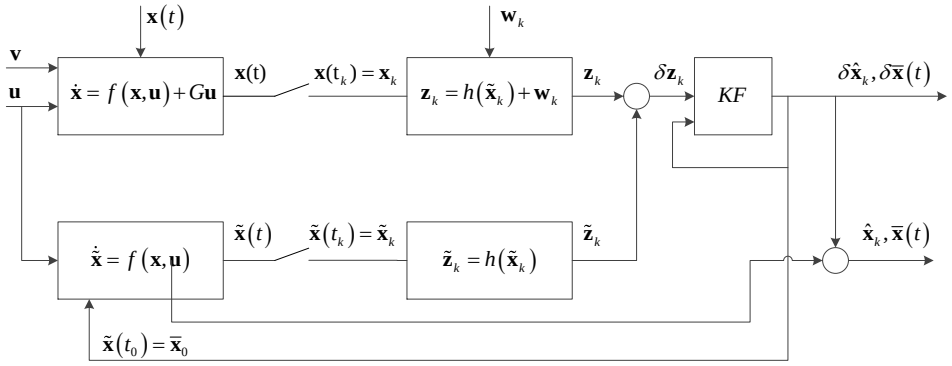
Figur 2-9 Veien fra verdenskoordinater til kamerakoordinater

3 MATEMATISK GRUNNLAG

Underkapitlene inneholder matematisk grunnlag for rapporten.

3.1 Utvidet Kalman Filter

Kalmanfilter er i utgangspunktet utledet for lineære systemer, men ved en utvidelse av filteret kan det også brukes for ulineære systemer. Filteret kalles da Utvidet Kalman Filter (UKF). Et blokkdiagram av et UKF vises i Figur 3-1.



Figur 3-1 Viser blokkdiagram av et UKF

Måleoppdateringslikningene for et UKF er:

$$\hat{\mathbf{x}}_k = \bar{\mathbf{x}}_k + K_k^*(\mathbf{z}_k - h_k^*(\bar{\mathbf{x}}_k)) \tag{3.1}$$

$$K_k^* = \bar{P}_k H_k^{*T}(H_k \bar{P}_k H_k^{*T} + R_k^*)^{-1} \tag{3.2}$$

$$\hat{P}_k = (I - K_k^* H_k^*)\bar{P}_k \tag{3.3}$$

Med tilbakekoblingen:

$$\hat{\mathbf{x}}_k^+ = \tilde{\mathbf{x}}_k + S^* \delta\hat{x}_k \tag{3.4}$$

$$\delta\hat{\mathbf{x}}_k^+ = (I - S^*)\delta\hat{x}_k \tag{3.5}$$

$$\hat{P}_k^+ = \hat{P}_k \tag{3.6}$$

Tidsoppdateringslikningene for et UKF er:

$$\dot{\bar{\mathbf{x}}} = \mathbf{f}^*(\bar{\mathbf{x}}, \mathbf{u}), \bar{\mathbf{x}}(\hat{t}_k) = \hat{\mathbf{x}}_k, \quad t \in [\hat{t}_k, t_{k+1}] \tag{3.7}$$

$$\dot{\bar{P}}(t) = F^*(t)\bar{P}(t) + \bar{P}(t)F^{*T}(t) + G^*(\mathrm{t}) \tag{3.8}$$

$$Q^*(t)G^{*T}(t) \tag{3.9}$$

$$\bar{P}(\hat{t}_k) = \hat{P}_k \tag{3.10}$$

3.2 RANSAC

RANSAC (Random Sample Consensus) er en metode for å estimere parametere i en matematisk modell for et datasett som inneholder "outliers". Resultatet av metoden kan

sammenliknes med minstekvadraters metode, men RANSAC vil gi et bedre estimat av parameterne ved et stort antall "outliers". Algoritmen ble først publisert av Fischler and Bolls i 1981 [6].

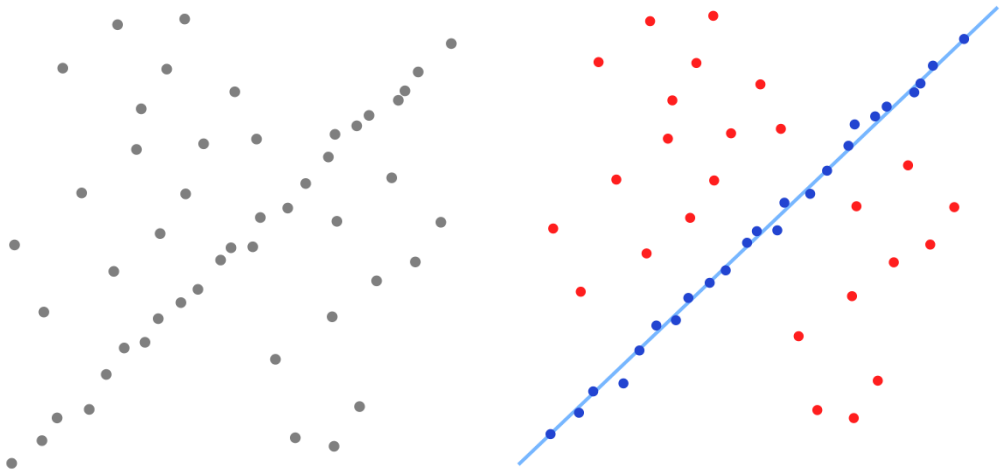
RANSAC er i hovedsak bygd opp av to steg som gjentar seg:

1. En undergruppe av et datasett som består av et minimum antall sampler blir tilfeldig valgt ut fra datasettet. En tilpasningsmodell med tilhørende parameter blir regnet ut ved hjelp av undergruppen.
2. Algoritmen sjekker deretter hvor mange av samplene i hele datasettet som ligger innenfor den utregnede tilpasningsmodellen pluss minus et avvik. Samplene som ligger innenfor tilpasningsmodellen pluss minus avviket kalles "inliers", samplene som ligger utenfor kalles "outliers".

Disse to stegene vil gjenta seg til minimum "inliers" er oppnådd, eller til et maks antall gjennomkjøringer er nådd og de parameterne som førte til flest "inliers" blir valgt.

3.2.1 Eksempel 1: 2D linjetilpassing

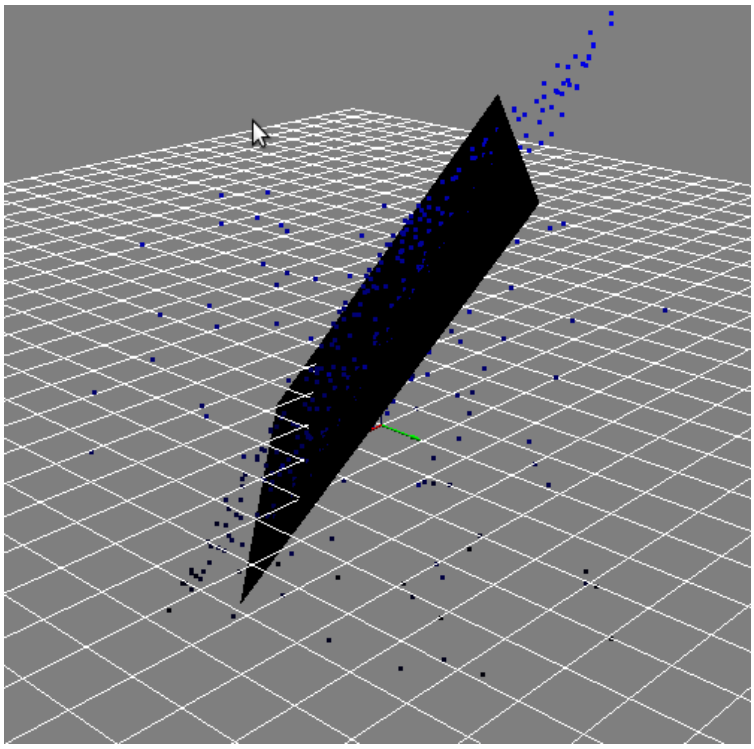
Det enkleste eksempelet er å tilpasse en linje i et 2D datasett. Undergruppen som blir valgt fra datasettet vil da inneholde 2 sampler. Figur 3-2 viser en linje tilpasset et datasett.



Figur 3-2 T.v. i figuren vises et datasett med "outliers" og "inliers", t.h. vises den modellen (en rettlinje) som passer datasettet best. Bilde er hentet fra [7].

3.2.2 Eksempel 2: 3D plantilpassing

Et annet eksempel er vist i Figur 3-3 som tilpasser ett plan i et 3D datasett. Undergruppen som blir valgt fra datasettet vil da inneholde 3 sampler.



Figur 3-3 Ett plan tilpasset et datasett ved bruk av RANSAC. Bildet er hentet fra [8].

3.3 Kvaternioner

Kvaternioner kan sees på som en romlig utvidelse av komplekse tall. Et sett av kvaternioner består av:

$$H = a \cdot 1 + bi + cj + dk \tag{3.11}$$

Hvor ijk har egenskapene med likheter til komplekse tall:

$$i^2 = j^2 = k^2 = ijk = -1 \tag{3.12}$$

Kvaternioner kan brukes til å representere orienteringer og rotasjoner av objekter i tre dimensjoner. Sammenlignet med Euler vinkler har kvaternioner noen fordeler. Det er lett å interpolere mellom to kvaternioner (som er nyttig for få jevne kamerabevegelser), de krever mindre regnekraft og unngå problemet med "gimbal locks" (dvs. singulariteter, f.eks. ved bruk av rull, pitch og yaw).

4 KAMERABASERT NAVIGASJON IMPLEMENTERT I MATLAB

Resten av rapporten bygger på et eksisterende kameranavigasjonsprogram som er programmert og utviklet i MATLAB av Javier Civera [1]. Programmet løser SLAM-problemet (Simultaneous Localization and Mapping), som vil si at det lokaliserer sin egenposisjon samtidig som den bygger opp et kart av omgivelsene.

Programmet bruker UKF og RANSAC for å estimere posisjon og orientering. RANSAC algoritmen bruker a priori sannsynligheter fra UKF for å velge ut en undergruppe i RANSAC algoritmen. Dette for å minimere antall iterasjoner i RANSAC. Vedlegg A skisserer hvordan UKF og RANSAC algoritmen fungerer. Programmet har en kapasitet på å estimere posisjonen i skala ved å følge rundt 100 bildemerker ved bilder tatt opp med 30fps i gangfart [9]. For å oppnå skala utnytter algoritmen kjennskapen om standardavviket til lineærakselerasjonen og vinkelakselerasjonen. Ifølge [1] som har implementert algoritmen i C++ skal algoritmen ha en Real-Time kapasitet på 60-70 bildemerker. Ved bruk av MATLAB har algoritmen en mye lavere hastighet. Avhengig av antall bildemerker vil en 1min lang video på 30fps ta fra 5-15timer. Kameranavigasjonsprogram har åpen kildekode og kan derfor modifiseres og bygges ut etter ønske [2].

Underkapitlene inneholder først en beskrivelse av originalprogrammet hvor informasjonen er hentet fra [9], og deretter en beskrivelse av hvordan programmet er endret slik at akselerometer- og gyroskopmålinger er implementert.

4.1 Matematisk modell for kameraets bevegelse

Kameraets tilstand $\mathbf{x}_v$ er satt sammen av posisjonen til kameraets optiske senter $\mathbf{r}^{WC}$, kameraets orientering er oppgitt i kvaternioner $\mathbf{q}^{WC}$, kameraets hastighet $\mathbf{v}^w$ relativt til verdensrammen F^W og vinkelhastigheten $\boldsymbol{\omega}^{C_k}$ relativt til kamerarammen F^C:

$$\mathbf{x}_{C_k}^W = \begin{bmatrix} \mathbf{r}_{C_k}^{WC} \\ \mathbf{q}_{k+1}^{WC} \\ \mathbf{v}^W \\ \boldsymbol{\omega}^W \end{bmatrix} \tag{4.1}$$

Programmet bruker en konstant lineær- og vinkelhastighets modell for å modellere kameraets bevegelse. Det antas at den lineære- og vinkelakselerasjonen a^W og α^C påvirker kameraet, som i hvert steg lager en lineær hastighetsinkrement $\mathbf{V}^W = \mathbf{a}^W \Delta t$, og vinkelhastigheten $\boldsymbol{\Omega}^W = \boldsymbol{\alpha}^W \Delta t$, med null i middelverdi og en kjent gaussisk fordeling. Det antas også at kovariansmatrisen er diagonal.

Tilstandsoppdateringen for modellen med konstant hastighets er:

$$\mathbf{f}_v = \begin{bmatrix} \mathbf{r}_k^{WC} \\ \mathbf{q}_{k+1}^{WC} \\ \mathbf{v}_{k+1}^W \\ \boldsymbol{\omega}_{k+1}^C \end{bmatrix} = \begin{bmatrix} \mathbf{r}_k^{WC} + (\mathbf{v}_k^W + \mathbf{V}_k^W)\Delta \mathrm{t} \\ \mathbf{q}_k^{WC} \times \mathbf{q}((\boldsymbol{\omega}_k^W + \boldsymbol{\Omega}^W)\Delta t) \\ \mathbf{v}_k^W + \mathbf{V}^W \\ \boldsymbol{\omega}_k^W + \boldsymbol{\Omega}^W \end{bmatrix} \tag{4.2}$$

hvor $\mathbf{q}((\boldsymbol{\omega}_k^W + \boldsymbol{\Omega}^W)\Delta t)$ er kvaternioner definert av rotasjonsvektoren $(\boldsymbol{\omega}_k^W + \boldsymbol{\Omega}^W)\Delta t$.

4.2 Euklids XYZ- parametrisering

XYZ parametrisering er når et landemerke er representert i euklidske koordinater.

$$\mathbf{x}_i = \begin{bmatrix} X_i & Y_i & Z_i \end{bmatrix} \tag{4.3}$$

4.3 Invers dybde-parametrisering

En annen måte å representere et landemerke på er å bruke invers dybde-parametrisering. I kameranavigasjonsprogrammet blir et 3D bildemerke i definert av en 6 dimensjonal vektor:

$$\mathbf{y}_i = \begin{bmatrix} x_i & y_i & z_i & \theta_i & \phi_i & \rho_i \end{bmatrix}, \tag{4.4}$$

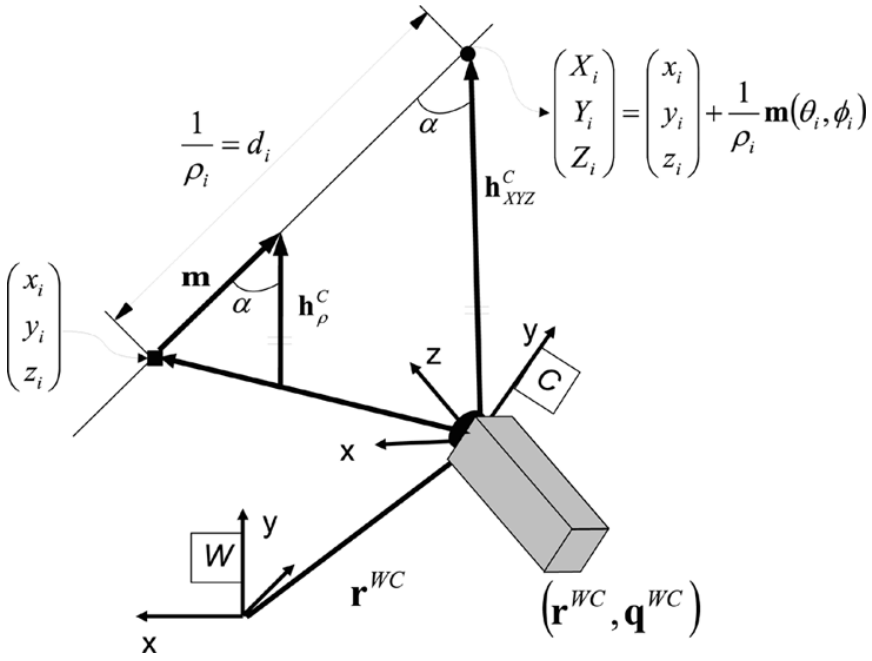
Landemerke $\mathbf{y}_i$ kan regnes om til XYZ 3D koordinater (se Figur 4-1):

$$\mathbf{x}_i = \begin{pmatrix} X_i \\ Y_i \\ Z_i \end{pmatrix} = \begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix} + \frac{1}{\rho_i}\mathbf{m}(\theta_i, \phi_i) \tag{4.5}$$

$$\mathbf{m} = (\cos\phi_i \sin\theta_i, -\sin\phi_i, \cos\phi_i \cos\theta_i)^T \tag{4.6}$$

Hvor $x_i \quad y_i \quad z_i$ er kameraets optiske senter hvor bildemerket først ble observert, oppgitt i verdenskoordinater. $\theta_i \quad \phi_i$ er asimut og elevasjon som bestemmer retningen til vektoren $\mathbf{m}$ fra kameraets optiske senter til bildemerket. ρ_i er definert som den inverse dybden til bildemerket $\rho_i = 1/d_i$, hvor d_i er dybden til bildemerket.

Fordelen ved å bruke invers dybde-parametrisering er at bildemerkene kan ha uendelig dybde og at det gir en bedre linearitet ved liten parallaksevinkel (når bildemerket har flyttet seg lite i bildeplanet fra bildemerket først ble observert, parallaksevinkel er vist som α i Figur 4-1). Bedre linearitet fører til at Kalman Filteret gir et bedre resultat. Ulempen ved invers dybde-parametrisering er at den krever den krever mer regnekraft. Derfor er programmet laget slik at når parallaksevinkelen er blitt stor nok vil den skifte til XYZ parametrisering.



Figur 4-1 Inverse dybde-parametrisering av et bildemerke [9].

4.4 Full tilstandsvektor

Hele tilstandsvektoren $\mathbf{x}$ er satt sammen av kameraets tilstandsvektor x_v og alle bildemerkene y_i:

$$\mathbf{x} = (x_v^T, y_1^T, y_2^T, ..., y_n^T)^T \tag{4.7}$$

4.5 Måleoppdateringsligningen

Hvert bildemerke fastsetter en begrensning mellom kamera lokasjonen og de tilhørende bildemerkene.

XYZ måleoppdateringsligningen:

$$\mathbf{h}^C = \mathbf{h}^C_{XYZ} = \mathbf{R}^{RW} \begin{pmatrix} X_i \\ Y_i \\ Z_i \end{pmatrix} - \mathbf{r}^{WC} \tag{4.8}$$

Invers dybde-parametrisering måleoppdatering:

$$\mathbf{h}^C = h^C_\rho = \mathbf{R}^{CW} \left(\rho_i \left(\begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} - \mathbf{r}^{WC} \right) \right) + \mathbf{m}(\theta_i, \phi_i) \tag{4.9}$$

4.6 Utvidelse av kameranavigasjonsprogrammet ved å implementere akselerometer- og gyroskopmålinger

Ved å måle lineære akselerasjonen og vinkelhastigheten med et akselerometeret og gyroskopet på en mobil (altså et lavkost akselerometer og gyroskop), er det ønskelig å se om det gir et bedre eller mer robust navigeringsresultat enn ved kun å bruke kameranavi-

gasjon. Det er også ønskelig å se om man kan redusere antall bildemerker men allikevel få et like godt navigasjonsresultat.

For å implementere akselerometer- og gyroskopmålinger må formel (4.2) endres slik at målingene blir et pådrag i tilstandsoppdateringen. Utledningen antar flat og ikke-roterende jord.

Den målte lineærakselerasjonen $\mathbf{a}_{k_m}^C \left[m/\mathrm{s}^2\right]$ og den målte vinkelhastigheten $\boldsymbol{\omega}_{k_m}^C$ $\left[rad/s\right]$ antas å ha en gaussisk støy med middelverdi $\bar{\mathbf{a}}_m$ og $\bar{\boldsymbol{\omega}}_m$. For å kun sitte igjen med de reelle målte verdiene må middelverdien trekkes fra $\mathbf{a}_{k_m}^C$ og $\boldsymbol{\omega}_{k_m}^C$:

$$\mathbf{a}_{k_{m-\bar{\mathbf{a}}_m}}^C = \mathbf{a}_{k_m}^C - \bar{\mathbf{a}}_m^C \tag{4.10}$$

$$\boldsymbol{\omega}_{k_{m-\bar{\mathbf{a}}_m}}^C = \boldsymbol{\omega}_{k_m}^C - \bar{\boldsymbol{\omega}}_m^C \tag{4.11}$$

Siden den matematiske modellen i (4.2) bruker lineærakselerasjonen og vinkelhastigheten oppgitt i verdenskoordinater må $\mathbf{a}_{k_{m-\bar{\mathbf{a}}_m}}^C$ og $\boldsymbol{\omega}_{k_{m-\bar{\mathbf{a}}_m}}^C$ først roteres til verdenskoordinater:

$$\mathbf{a}_{k_m}^W = R_C^W \mathbf{a}_{k_{m-\bar{\mathbf{a}}_m}}^C \tag{4.12}$$

$$\boldsymbol{\omega}_{k_m}^W = R_C^W \boldsymbol{\omega}_{k_{m-\bar{\mathbf{a}}_m}}^C \tag{4.13}$$

Den målte lineærakselerasjonen $\mathbf{a}_{k_m}^W$ inneholder gravitasjonen $\mathbf{g}^W$ som må trekkes fra:

$$\mathbf{a}_{k_{m-\mathbf{g}}}^W = \mathbf{a}_{k_m}^W - \mathbf{g}^W = \mathbf{a}_{k_m}^W - \begin{bmatrix} 0 \\ 0 \\ 9.81 \end{bmatrix} \tag{4.14}$$

For hvert tidskritt produserer den målte lineærakselerasjonen et lineært hastighetsinkrement:

$$\mathbf{V}_{k_m}^W = \mathbf{a}_{k_{m-g}}^W \Delta t \tag{4.15}$$

Vinkelhastigheten $\boldsymbol{\omega}_{k_m}^W$ er målt direkte og vil da erstatte $\boldsymbol{\omega}_k^W + \boldsymbol{\Omega}^W$ i (4.2).

Ved bruk av $\mathbf{V}_{k_m}^W$ og $\boldsymbol{\omega}_{k_m}^W$ vil tilstandsoppdateringen oppgitt i (4.2) bli endret til:

$$\mathbf{f}_v = \begin{bmatrix} \mathbf{r}_k^{WC} \\ \mathbf{q}_{k+1}^{WC} \\ \mathbf{v}_{k+1}^W \\ \boldsymbol{\omega}_{k+1}^C \end{bmatrix} = \begin{bmatrix} \mathbf{r}_k^{WC} + (\mathbf{v}_k^w + \mathbf{V}_{k_m}^W)\Delta \mathrm{t} \\ \mathbf{q}_k^{WC} \times \mathbf{q}(\omega_{k_m}^W \Delta t) \\ \mathbf{v}_k^W + \mathbf{V}_{k_m}^W \\ \boldsymbol{\omega}_{k_m}^W \end{bmatrix} \tag{4.16}$$

5 TESTOPPSETT OG FORBREDELSER FOR EKSEPRIMENT MED BILDESEKVENS FRA EGET KAMERA OG MÅLTE TREGHETSDATA

For å gjennomføre et eksperiment med kameranavigasjon med og uten målte treghetsdata er det flere forberedelser som må gjøres. Underkapitlene inneholder oppsett og forberedelser som ble gjort i forkant av eksperimentet.

5.1 Testrigg

For å redusere støy på kamera og treghetsplattformen ble det laget en testrigg som vises i Figur 5-1. Testriggen består av et kamera, en mobiltelefon (treghetsplattform), en stoltralle og en verktøykasse. Stoltrallen kan enkelt trilles rundt i en jevn og rolig bevegelse. En tung verktøykasse er plassert slik at riggen får et lavt tyngdepunkt som vil medføre en ytterligere jevn og rolig bevegelse av kameraet og treghetsplattformen.

Figur 5-1 Viser testriggen som består av kamera, mobiltelefon (treghetsplattform), stoltralle og en verktøykasse.

5.2 Kamera

Det var ønskelig å bruke en mobiltelefon til både å ta opp video og treghetsdata under eksperimentet. Men på grunn av mobiltelefonen ikke kunne filme samtidig som den gjorde treghetsdata opptak ble det brukt et ordinært digitalkamera i tillegg. Kameraet som ble brukt er et Canon IXUS 220 HS. Spesifikasjonene på kameraet er hentet fra kameraets manual [10] og er vist i Tabell 5-1.

Tabell 5-1 Viser data på kamerat som er brukt

Navn	Canon IXUS 220 HS
Megapiksler	12,1
Maks oppløsing	4000 x 3000 piksler
Fokallengde	4,3-21,5mm
Sensor type	CMOS
Sensorstørrelse	6,16 x 4,62mm
Pikselstørrelse	1,54um
Data type video	MOV
Videooppløsing brukt	320 x 240 og 640 x 480
Opptakshastighet	30fps

Kameraet ble brukt uten zoom, fokal lengen vil derfor være konstant 4,3mm

5.3 Kamerakalibrering

Kamerakalibrering er nødvending for å få finne de eksakte kameraparameterne til det spesifikke kameraet som brukes. Kameraprodusenten oppgir kameraparameterne, men disse kan variere fra kamera til kamera. Dette kan f.eks. skyldes unøyaktighet i montering av bildesensoren. Det er derfor nødvending å kalibrere kameraet for å få de riktige parameterne. Kalibreringsalgoritmen vil også gi ut parametere for kompensasjon av forvrengningen i linsen.

Kalibreringsverktøyet som er brukt er Camera Calibration Toolbox for MATLAB og laget av Jean-Yves Bouguet [11].

Fremgangsmetoden for bruk av kalibreringsverktøyet er:

1. Lage en kalibreringsrigg med et plant sjakkmønster (se Figur 5-2)
2. Ta bilder av kalibreringsriggen
3. Importere bildene til MATLAB kalibreringsverktøyet
4. Merk av de fire ytterste hjørnene i sjakkbrettmønsteret
5. Start kalibrering
6. Analysere feilen
7. Redefinere avmerkede hjørner om nødvendig
8. Start kalibrering på nytt
9. Les ut kalibreringsverdiene

Figur 5-2 Kalibreringsriggen som ble brukt for å kalibrere kameraet. Kvadratene i rutemønstret er 30x30mm.

De aktuelle kalibreringsparametere fra Bouguet's kalibreringsverktøyet er presentert i Tabell 2-2. Bouguet's kameramodell bruker 5 parametere for å representere forvrengningen i linsen.

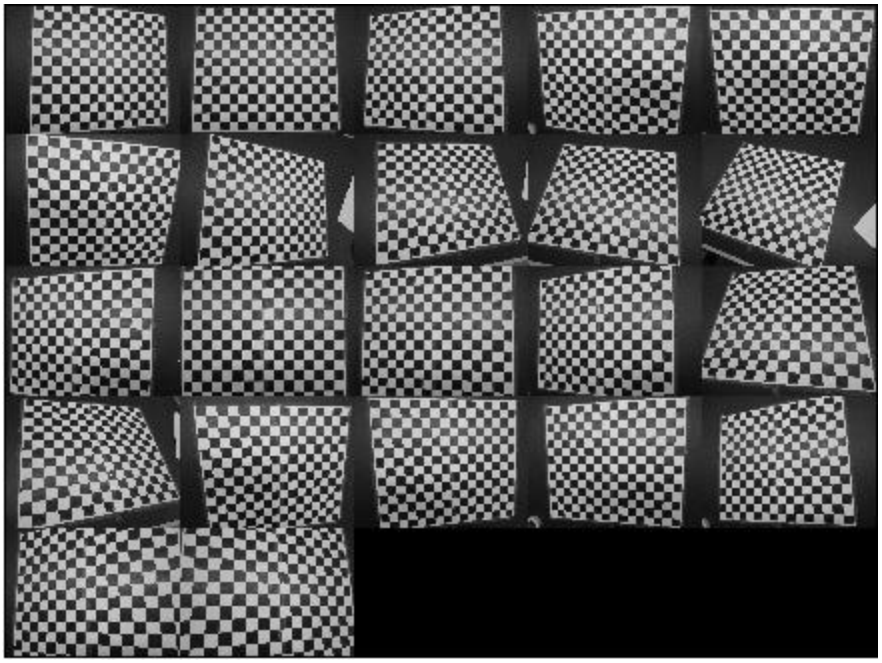
Tabell 5-2 Parameterne i Bouguets kalibreringsrutinen

Parameter	**Rapportens notasjon**	**Enhet**	**Beskrivelse**
fc	$f_{uv} = \begin{bmatrix} f_u \\ f_v \end{bmatrix}$	$[piksl]$	Fokallengden
cc	$O_{uv} = \begin{bmatrix} O_u \\ O_v \end{bmatrix}$	$[piksl]$	Bildesenteret
kc	$k_c = \begin{bmatrix} k_c1 \\ k_c2 \\ k_c3 \\ k_c4 \\ k_c5 \end{bmatrix}$	-	Forvrengning
alpha_c	α_c	-	Skjevkoeffisient

Som Tabell 5-2 viser at Bouguet bruker en 5x1 vektor for å beskrive forvrengningen i linsen[11]. MATLAB kameranavigasjonsprogrammet [2] som brukes i dette prosjektet bruker en Tsai forvrengingensmodell. Denne beskriver forvrengingen i lisen ved bruk av en 2x1 matrise. Det er derfor nødvendig å konvertere Bouguets kameramodellparametere til Tsai forvrengingensmodell-parametere som gjøres i Kapittel 5.3.2.

5.3.1 Resultat av Bouguet's kalibreringsalgoritme

Det ble brukt 22 bilder av kalibreringsriggen for å kalibrere kameraet. Bildene som ble brukt er vist i Figur 5-3, og resultatet av kalibreringen er presentert i Tabell 5-3.



Figur 5-3 Viser bildene av kalibreringsriggen som ble brukt i kalibreringsverktøyet

Tabell 5-3 Viser Bouguet kalibrerte kamera parametere

Parameter	**Resultat 4000x3000**	**Omregnet resultat til 320x240**
$f_{uv} = \begin{bmatrix} f_u \\ f_v \end{bmatrix}$	$\begin{bmatrix} 2802.9 \\ 2804.1 \end{bmatrix}$	$\begin{bmatrix} 224.2 \\ 224.3 \end{bmatrix}$
$O_{uv} = \begin{bmatrix} O_u \\ O_v \end{bmatrix}$	$\begin{bmatrix} 1974.8 \\ 15890.0 \end{bmatrix}$	$\begin{bmatrix} 159.4213 \\ 129.1688 \end{bmatrix}$
$k_c = \begin{bmatrix} k_c1 \\ k_c2 \\ k_c3 \\ k_c4 \\ k_c5 \end{bmatrix}$	$\begin{bmatrix} -0.03514 \\ 0.01485 \\ 0.00871 \\ 0.0005 \\ 0.00000 \end{bmatrix}$	$\begin{bmatrix} -0.03514 \\ 0.01485 \\ 0.00871 \\ 0.0005 \\ 0.00000 \end{bmatrix}$
$alpha\_c$	$[0]$	$[0]$

Kameranavigasjonsprogrammet bruker fokallengden oppgitt i mm. Derfor må fokallengden i Tabell 5-3 regnes om til mm. Det kan gjøres med formel (5.1).

$$f_{xy} = \begin{bmatrix} f_x \\ f_y \end{bmatrix} = \begin{bmatrix} \frac{d_b f_u}{rader} \\ \frac{d_l f_v}{kolonner} \end{bmatrix} \tag{5.1}$$

Hvor d_w og d_l er bildesensor størrelsen i bredde og lengde oppgitt mm. *rader* og *kolonner* er bildeoppløsningen i antall piksler.

5.3.2 Konvertering av Bouguet's kameramodell til Tsai

Bouguet's kameraalgoritme bruker en kameramodell med 5 forvrengningsparametere. Kameranavigasjonsprogrammet bruker en Tsai modell for å beskrive linseforvrengingen. [2] har laget en overgangs algoritme som konverterer Bouguet's forvrengingsparameteren til Tsai parametere. Tsai parameterne er vist i Tabell 5-4.

Tabell 5-4 Kalibrerte Tsai kameraparametere

Parameter	**Resultat**
$k_c = \begin{bmatrix} k_c 1 \\ k_c 2 \end{bmatrix}$	$\begin{bmatrix} 0.0010524 \\ 3.0398895e-05 \end{bmatrix}$

5.3.3 Kalibrerte kameraparametere

Tabell 5 Kameraparameterne som ble brukt

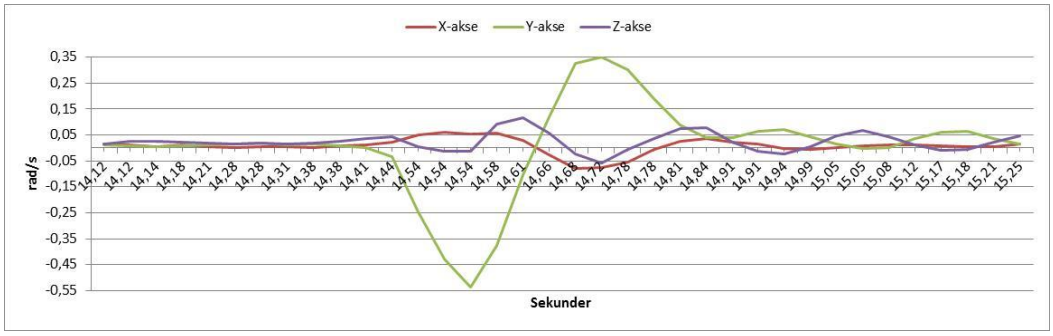
Parameter	**Enhet**	**Verdi**
$f_{xy} = \begin{bmatrix} f_x \\ f_y \end{bmatrix}$	$[mm]$	$\begin{bmatrix} 4.315 \\ 4.317 \end{bmatrix}$
$O_{uv} = \begin{bmatrix} O_u \\ O_v \end{bmatrix}$	$[piksel]$	$\begin{bmatrix} 159.4213 \\ 129.1688 \end{bmatrix}$
$k_c = \begin{bmatrix} k_c 1 \\ k_c 2 \end{bmatrix}$	-	$\begin{bmatrix} 0.0010524 \\ 3.0398895e-05 \end{bmatrix}$

5.4 Treghetssensor

En Samsung Note II med applikasjonen Sensor Kinetics Pro ble brukt som treghetssensor. Sensor Kinetics Pro kan lagre akselerometer- og gyromålinger som CSV fil som kan importeres inn i MATLAB.

5.5 Synkronisering av akselerometer- og gyromålinger mot bilder

Synkronisering av akselerometer- og gyromålinger mot bildesekvensen ble gjort visuelt på måledataen. Under starten av opptaket ble det gjort en karakteristisk yaw bevegelse med kamera og treghetsplattformen. Denne bevegelsen er lett å trekke ut fra gyromålingene og bildene. Siden akselerometer- og gyromålingene har samme tidsstempel vil akselerometermålingene ha samme starttidspunkt som gyromålingen.



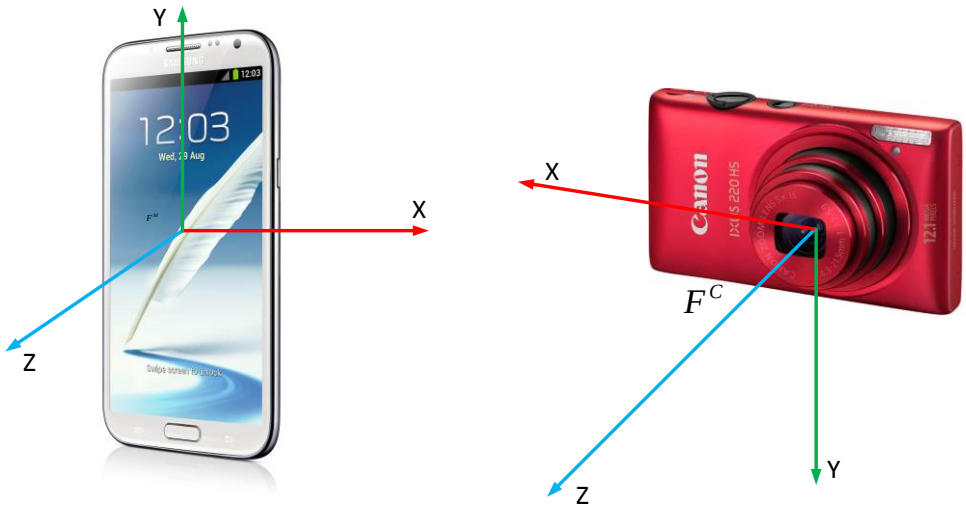
Figur 5-4 Viser gyromålingene ved en karakteristisk yaw bevegelse for å synkroniseres med bildeserien. Ved ca. 14,84 sekunder ble målingene synkronisert ved å finne samme bevegelsen i kamerabildet.

5.6 Koordinatsystemene

Figur 5-5 viser at kameraet og treghetssensorene har forskjellige koordinatsystem. Ved importering av treghetssensordataen til MATLAB må det bli utførts en koordinattrans-formasjon av akselerometer- og gyromålingene:

$$\begin{aligned} \mathbf{a}_{k_m}^C &= \mathbf{R}_M^C \mathbf{a}_{k_m}^M = [-a_{k_m x}^M \quad -a_{k_m y}^M \quad a_{k_m z}^M]^T \\ \boldsymbol{\omega}_{k_m}^C &= \mathbf{R}_M^C \boldsymbol{\omega}_{k_m}^M = [-\omega_{k_m x}^M \quad -\omega_{k_m y}^M \quad \omega_{k_m z}^M]^T \end{aligned} \tag{5.2}$$

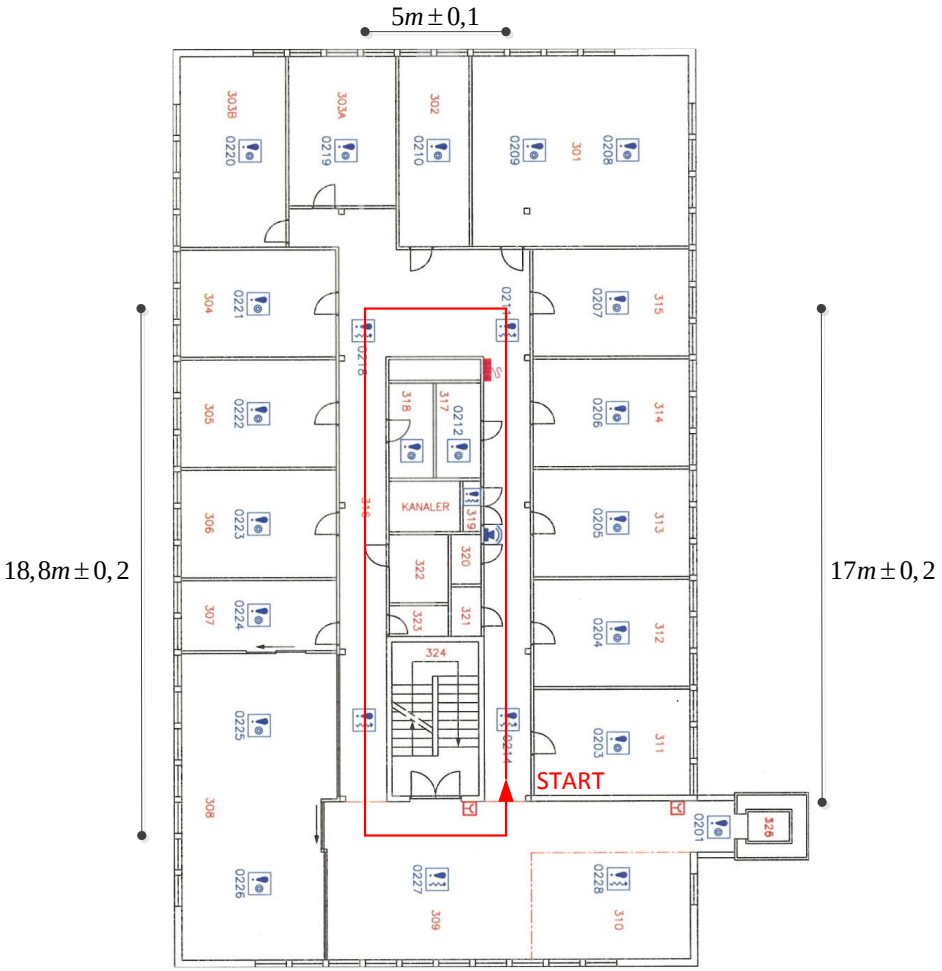
Hvor $\mathbf{a}_{k_m}^M$ og $\boldsymbol{\omega}_{k_m}^M$ er akselerasjon- og vinkelhastighetsmåling representert i treghets-sensorrammen, $\mathbf{a}_{k_m}^C$ og $\boldsymbol{\omega}_{k_m}^C$ er akselerasjon- og vinkelhastighetsmåling rotert til kamerarammen.



Figur 5-5 Koordinatsystemet til treghetssensorene og kameraet.

6 RESULTAT AV EKSTPRIMENT MED BILDESEKVENS FRA EGET KAMERA OG MÅLT TREGHETSSENSOR DATA

Eksperimentet ble utført i 3. etasje på UNIK. Ruten som ble brukt er vist i Figur 6-1.



Figur 6-1 Eksperimentet ble utført i 3. etasje på UNIK. Ruten som ble gått er merket med rød strek. Ruten er tilnærmet 46m lang. Rød pil indikerer hvor opptaket startet.

Opptaket ble gjort i oppløsningen 640x480 med 30fps. Videoen ble deretter etterprosessert i programmet VLC media player for å nedskalere videoen til 320x240 (for å redusere prosesseringstid i MATLAB) og for å lagre hvert bilde i videoen til bildeformatet PGM. Videoen er 98 sekunder lang og består dermed av 2940 bilder. Dette vil gi tilnærmet 64 bilder pr. meter.

Eksperimentene er utført ved å følge et ulikt antall bildemerker. Ved å følge ett gitt antall bildemerker kan det forkomme at programmet bruker både færre eller flere bildemerker underveis. Dette skyldes at hvis bildegjenkjenningsalgoritmen ikke finner korrelasjon av bildemerket i neste bilde blir bildemerket tatt ut av antall "fulgte bildemerker" og

programmet vil da lete etter et nytt bildemerke for å opprettholde 35 bildemerker. Gitt neste iterasjon, og bildegjenkjenningsalgoritmen finner korrelasjon på alle 35 aktive bildemerker, og i tillegg finner korrelasjon på bildemerket som ble deaktivert i forrige iterasjon, vil programmet bruke 36 bildemerker denne iterasjonen. I tilfellet der algoritmen bruker færre bildemerker enn det gitte antall, er dette på grunn av at algoritmen ikke klarer å finne nok bildemerker i bildet.

6.1 Kamera-navigasjonseksperiment med egne bilder

Underkapitlene viser resultater fra eksperimentet med en bildeserie tatt med eget kamera. Eksperimentet ble utført ved å kjøre kameranavigasjonsprogrammet med 15, 25, 35, 50 og 100 bilder. Resultatet viser at programmet gir et uforutsigbart estimat av posisjonen. Ved å følge 15 bildemerker gir programmet best resultat / nærmest skala, men etter halvparten av ruten drifter posisjonen ut. Først ved å følge 100 bildemerker blir ruten relativt riktig, men uten skala.

Resultatet av eksperimentene er presentert i figurer. En forklaring av hva figurene består av er gitt her:

Forklaring av innholdet i vestre del figurene:

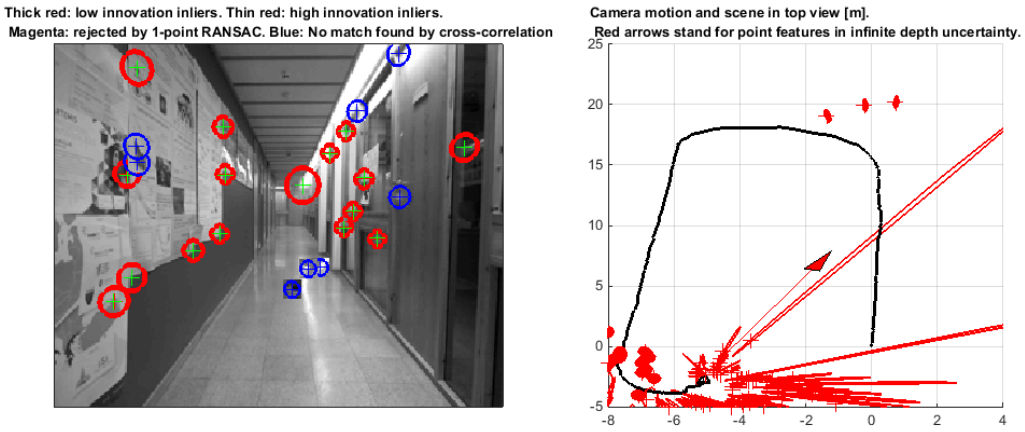
- Rød ring markerer bildemerker som er med i algoritmen.
- Blå ringer markerer at ingen match er funnet ved krysskorrelasjon av bildemerket fra forrige bilde. Bildemerket er da tatt ut som aktivt bildemerke.
- Lilla ring viser bildemerker tatt ut som aktivt bildemerke av RANSAC algoritmen.

Forklaring av innholdet i vestre del figurene:

- Svart strek viser estimert posisjons trajektor.
- Rød pil viser bildemerke med uendelig dybde.
- Rød ellipse viser usikkerheten til bildemerket.

6.1.1 Kamera-navigasjonseksperiment med 15 bildemerker

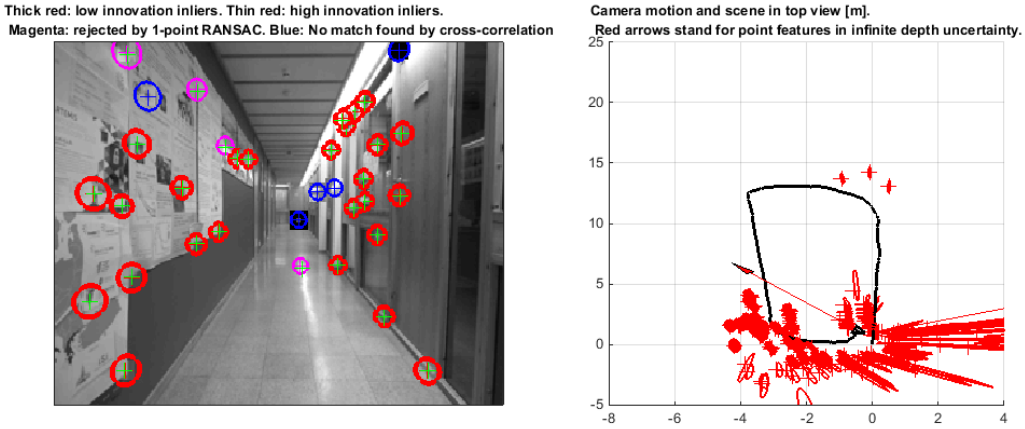
Eksperimentet gir godt estimat av skala av ruten, men etter halvveis drifter skala og posisjonsestimat ut. Resultatet er vist i Figur 6-2.



Figur 6-2 Kameranavigasjonsprogrammet med minimum 15 bildemerker

6.1.2 Kamera-navigasjonseksperiment med 25 bildemerker

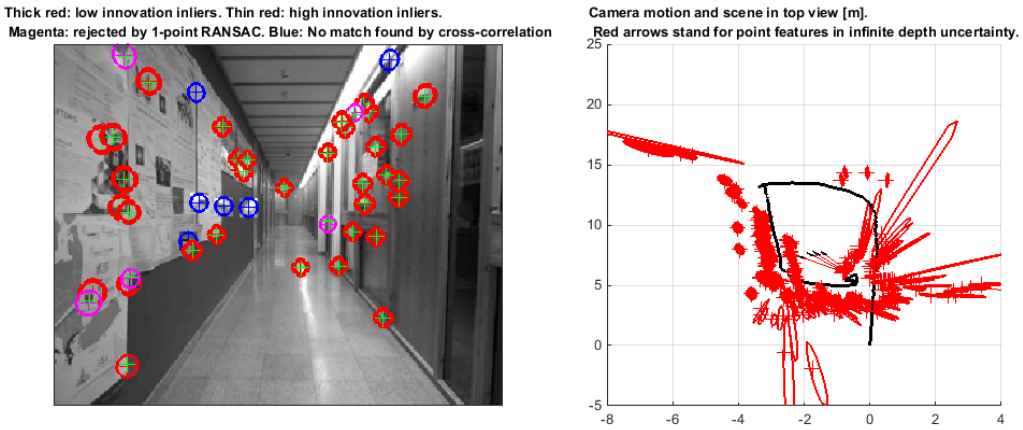
Eksperimentet gir et relativt godt estimat av ruten, men skala blir feil. Resultatet er vist i Figur 6-3



Figur 6-3 Kameranavigasjonsprogrammet med minimum 25 bildemerker

6.1.3 Kamera-navigasjonseksperiment med 35 bildemerker

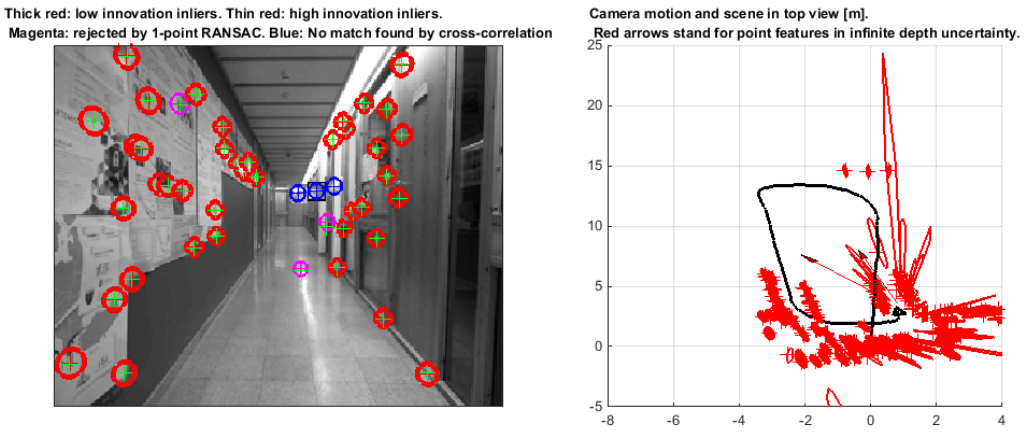
Eksperimentet gir et dårlig estimat av posisjonen og skala. Resultatet er vist i Figur 6-4.



Figur 6-4 Kameranavigasjonsprogrammet med minimum 35 bildemerker

6.1.4 Kamera-navigasjonseksperiment med 50 bildemerker

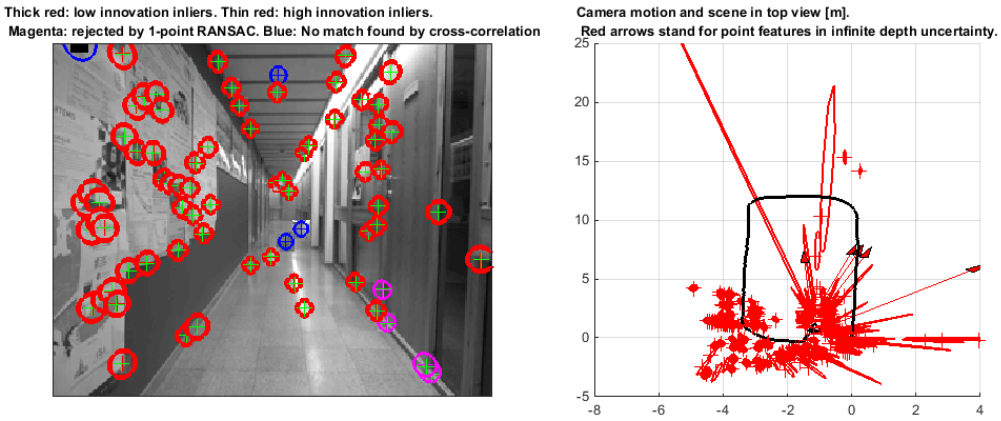
Eksperimentet gir et dårlig estimat av posisjonen og skala. Resultatet er vist i Figur 6-5.



Figur 6-5 Kameranavigasjonsprogrammet med minimum 50 bildemerker

6.1.5 Kamera-navigasjonseksperiment med 100 bildemerker

Eksperimentet gir et godt estimat av posisjonen men feil skala. Resultatet er vist i Figur 6-6.



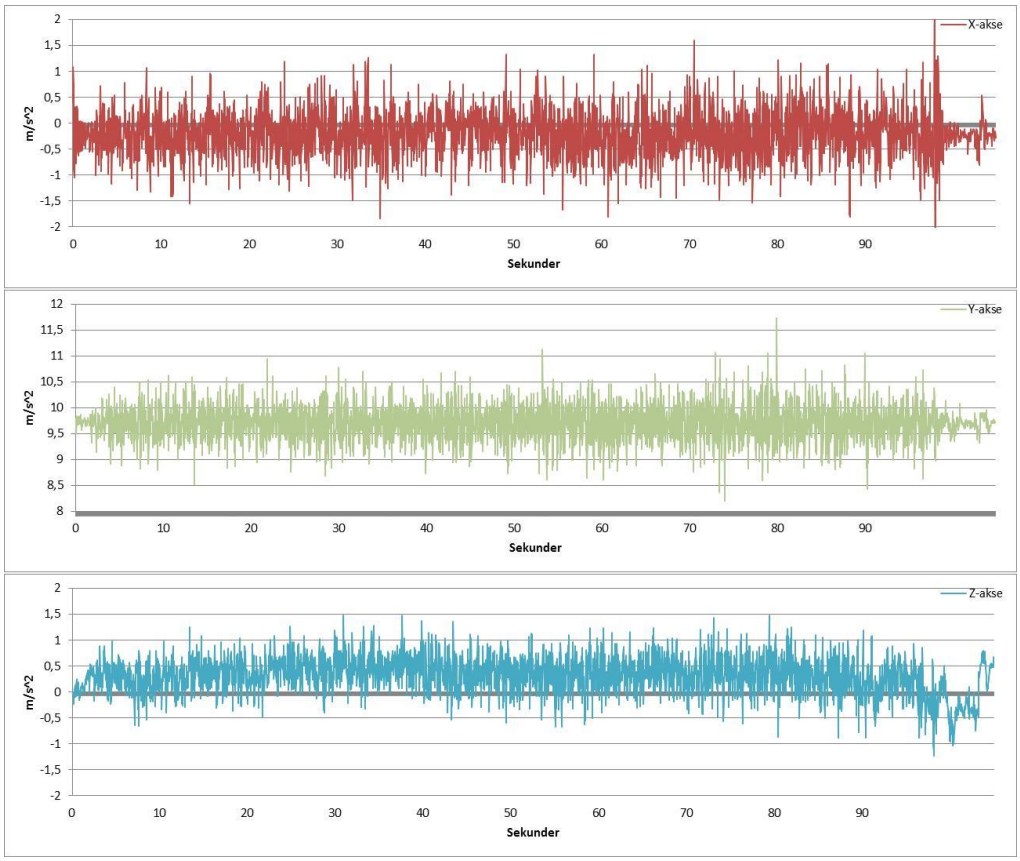
Figur 6-6 Kameranavigasjonsprogrammet med minimum 50 bildemerker

6.2 Kamera-navigasjonseksperiment med egne bilder og akselerometer- og gyromålinger

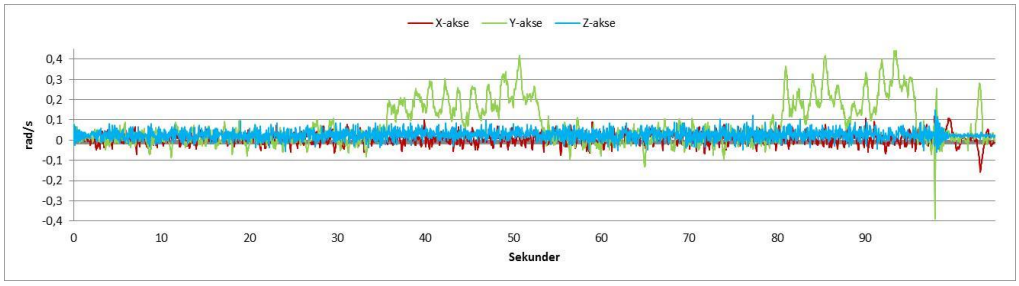
Eksperimentet ble utført med samme bildesekvens som i eksperimentet i Kapittel 6.1.

6.2.1 Opptak av akselerometer- og gyromålinger treghets-sensorene

Akselerasjons- og gyromålingene er vist i henholdsvis Figur 6-7 og Figur 6-8. Målingene ble tatt opp i 30 Hz og ble tatt opp samtidig som bildeserien. Figuren viser at akselerasjonsmålingene har et høyt innhold av støy og har en lav SNR (Signal-to-Noise ratio).



Figur 6-7 Akselerometermålinger tatt opp i 30 Hz med en Samsung Note II mobiltelefon



Figur 6-8 Gyroskopmålingene tatt opp i 30 Hz med en Samsung Note II mobiltelefon

Middelverdien og standardavviket til treghetssensorene ble regnet ut med henholdsvis formel (6.1) og (6.2) på et opptak med konstant hastighet og ingen rotasjon.

$$\bar{x} = \frac{\sum_{i=1}^{N} x_i}{N} \tag{6.1}$$

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2} \tag{6.2}$$

Utregning av standardavviket og middelverdien for akselerometeret gir:

$$\begin{aligned} \boldsymbol{\sigma}_{\mathbf{a}} &\approx 0.35 \left[m/s^2 \right] \\ \bar{\mathbf{a}} &\approx [0.2348\ 0.0789\ 0.3062]^T \left[m/s^2 \right] \end{aligned} \tag{6.3}$$

og for gyroskopet:

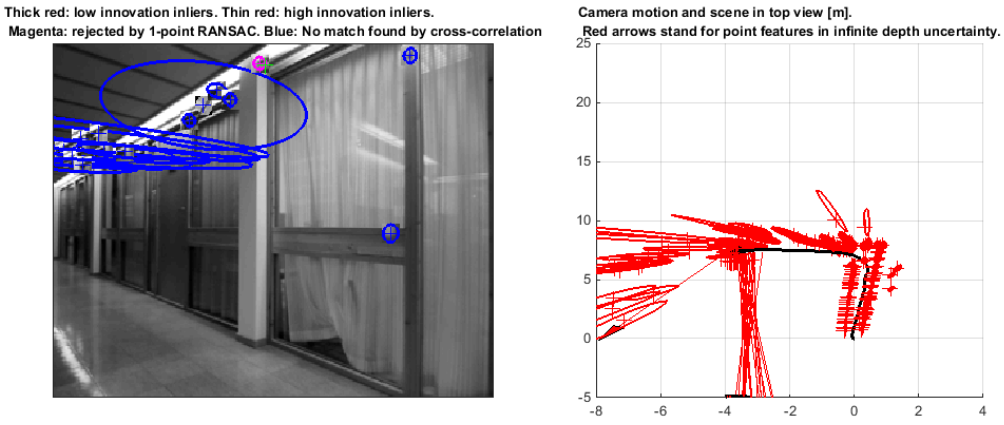
$$\begin{aligned} \boldsymbol{\sigma}_g &\approx 0,75 \ \left[rad / s \right] \\ \overline{\boldsymbol{\omega}} &\approx [\text{-}0.0090 \ \text{-}0.0117 \ 0.0234]^T \ \left[rad / \text{s} \right] \end{aligned} \tag{6.4}$$

6.2.2 Kamera-navigasjonseksperiment med 25 og 100 bildemerker med utregnet standardavvik og middelverdi

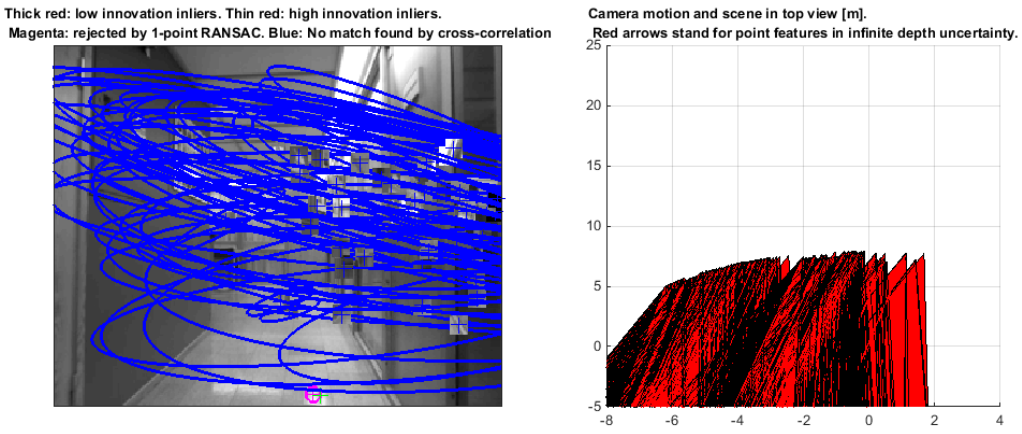
Figur 6-9 og Figur 6-10 viser resultater fra kameranavigasjonsprogrammet med bruk av gyro- og akselerometermålinger ved henholdsvis å følge 25 og 100 bildemerker. Figurene viser et dårlig navigasjonsresultat. Dette skyldes i hovedsak to ting:

1. Grunnen til det dårlige estimatet av skalaen frem til første sving er på grunn av akselerometer målingene inneholder lite informasjon om akselerasjonen og inneholder mye støy. Det vil derfor være en løsning å sette opp standardavviket, slik at man "stoler" mindre på målingene. Standardavviket til akselerasjonsmålingene ble juster til 2 $\left[m / s^2 \right]$. Standardavviket til gyroskopet ble holdt uendret.
2. Grunnen til at estimatet drifter ut etter første sving er at ved kun en liten feil i den estimerte orienteringen til kameraet vil fører til problemer når akselerasjonsdataen fra akselerometeret skal roteres til verdenskoordinater før prosessering. Feilestimeringen fører til at målt akselerasjon som inneholder gravitasjon **g** blir rotert slik at **g** ikke blir stående nedover (noe den gjør siden treghetsplattformen ikke har noen bevegelse i pitch og roll). Gravitasjonen vil derfor fordele seg på andre akser og vil gi feile hastigheter som igjen fører til feil posisjon. Dette kan løses/forbedres på ulike måter. En metode kan være implementere 3-akset magnetometermålinger i måleoppdateringen og vil dermed få et bedre estimat av orienteringen. En annen måte kan være å utnytte informasjon som er kjent for systemet. Kjent informasjon i eksperimentet i denne rapporten er at det kun vil opptre en vinkelhastighet rundt z-aksen. Utfra dette kan det lages en syntetisk magnetometermåling som kan brukes i måleoppdateringen.

at orienteringen av kameraet blir feil-estimert, som igjen fører til at Det oppstår problemer



Figur 6-9 Kameranavigasjonsprogrammet med minimum 25 bildemerker og akselerometer- og gyromålinger



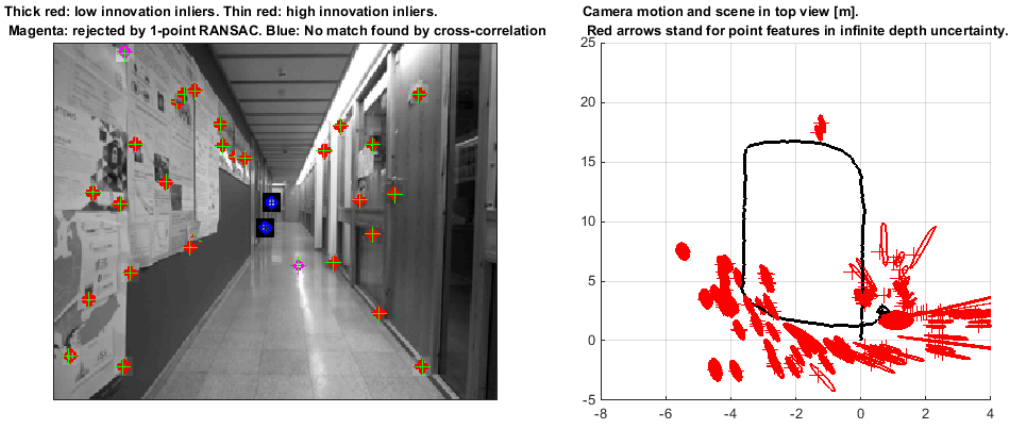
Figur 6-10 Kameranavigasjonsprogrammet med minimum 100 bildemerker og akselerometer- og gyromålinger

6.2.3 Kamera-navigasjonseksperiment med 15 bildemerker med justert standardavvik og middelverdi

Videre eksperimenter bruker:

$$\begin{aligned} \boldsymbol{\sigma}_{\mathbf{a}} &= 2 \ \left[m/s^2 \right] \\ \boldsymbol{\sigma}_{g} &= 0,75 \ \left[m/s^2 \right] \end{aligned} \tag{6.5}$$

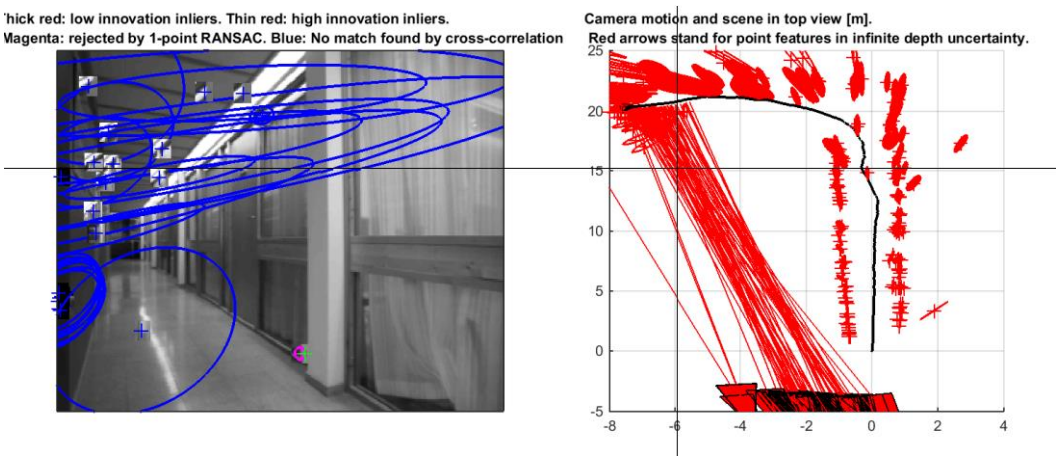
Eksperimentet gir en god estimering av posisjonen selv med bare 15 bildemerker. Dette viser potensialet ved å implementere gyro- og akselerometermålinger i kamerabasert navigasjon.



Figur 6-11 Kameranavigasjonsprogrammet med minimum 15 bildemerker og akselerometer- og gyromålinger

6.2.4 Kamera-navigasjonseksperiment med 25 bildemerker med justert standardavvik og middelverdi

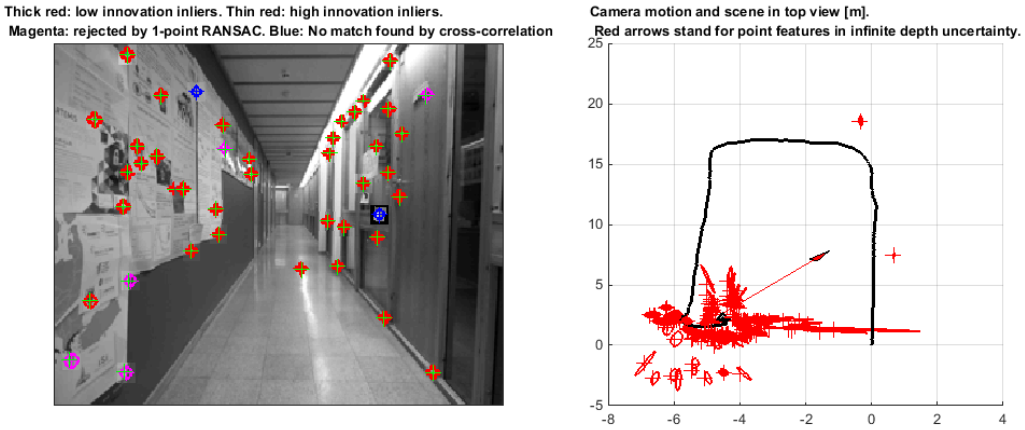
Eksperimentet gir et dårlig resultat som skyldes feil i orienteringsestimatet forklart i 2. punkt i kapittel 6.2.2.



Figur 6-12 Kameranavigasjonsprogrammet med minimum 25 bildemerker og akselerometer- og gyromålinger

6.2.5 Kamera-navigasjonseksperiment med 35 bildemerker med justert standardavvik og middelverdi

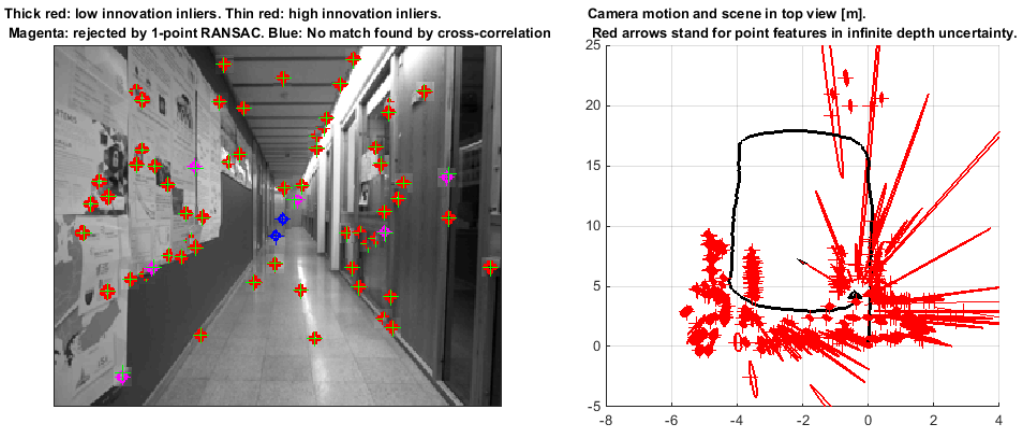
Eksperimentet gir et veldig godt estimat frem til siste sving som skyldes feil i orienteringsestimatet forklart i 2. punkt i kapittel 6.2.2.



Figur 6-13 Kameranavigasjonsprogrammet med minimum 35 bildemerker og akselerometer- og gyromålinger

6.2.6 Kamera-navigasjonseksperiment med 50 bildemerker med justert standardavvik og middelverdi

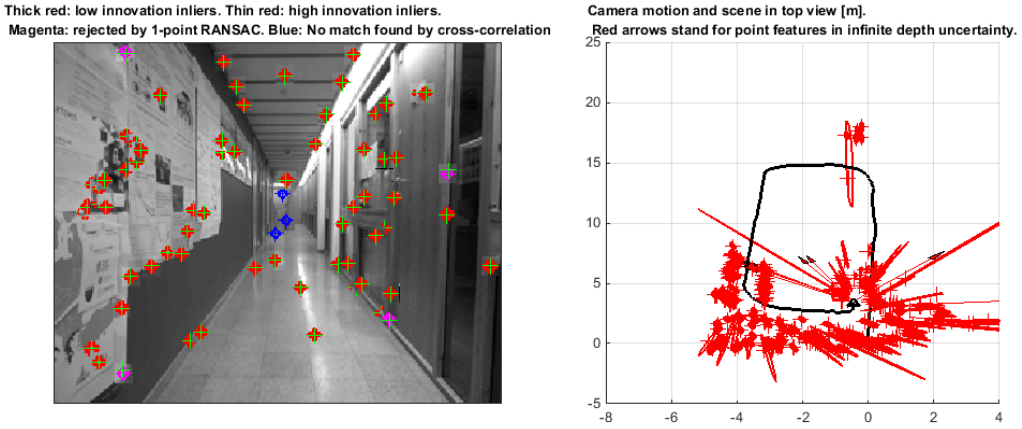
Eksperimentet gir et godt estimat av med noe feilestimert posisjon under siste halvdel. Resultatet er vist i Figur 6-14.



Figur 6-14 Kameranavigasjonsprogrammet med minimum 75 bildemerker og akselerometer- og gyromålinger

6.2.7 Kamera-navigasjonseksperiment med 100 bildemerker med justert standardavvik og middelverdi

Eksperimentet gir et dårligere resultat enn ved å følge færre bildemerker. Dette kan skyldes feil i orienteringsestimatet forklart i kapittel 6.2.2 og at programmet blir "sikrere" på målingene fra kamera. Målingene fra kamera vist i kapittel 6.1.5 har ikke riktig skala og vil dermed bidra til dårligere estimering av posisjonen. Resultatet er vist i Figur 6-15.



Figur 6-15 Kameranavigasjonsprogrammet med minimum 100 bildemerker og akselerometer- og gyromålinger

7 DISKUSJON

Kameranavigasjon med egne bilder fungerer bra med et høyt (≈ 100) antall bildemerker, men ikke optimalt siden skalaen blir feil. Dette kan skyldes feil i kameraparameterne fra kalibreringsalgoritmen eller feil antagelser om støy på lineær- og vinkelakselerasjon.

For å oppnå best mulig gyroskop- og akselerometermålinger ble det satt sammen en rigg for å feste kamera og treghetsplattformen. Det ble en stor forbedring på målingene ved bruk av riggen, men fortsatt mye bevegelsesstøy spesielt på akselerasjonen.

Gyro- og akselerasjonsmålingen ble visuelt synkronisert med kamerabildene ved å plotte gyrodata og se etter en karakteristikk yaw-bevegelse som ble utført i starten opptak. Ved å finne den samme bevegelsen i kamerabildene ble tiden synkronisert. Synkroniseringsmetoden blir vurdert som god nok for eksperimentet, med en nøyaktighet på rundt $\pm$ ett sample eller $\pm$ 33ms.

Gyro- og akselerasjonsmålingen ble tatt opp i 30Hz som er samme frekvensen som måleoppdateringen (30fps). Det ville med fordel være lurt å ta opp gyro- og akselerasjonsmålingen med en høyere frekvens og dermed kjøre tidsoppdateringen med en høyere frekvens. Dette ville ført til at noe av støyen på målingene blir midlet bort.

Støyen til gyro- og akselerasjonsmålingen blir sett på som hvit og gaussisk. Det blir også antatt en flat ikke-roterende jord. Dette er grove antagelser, men vil ha neglisjerbar betydninger på eksperimentene som er utført i denne rapporten.

Implementering av gyro- og akselerometermålinger gir en stor forbedring av navigasjonsresultatet ved at posisjon og orientering estimeres med en mindre feil enn ved ren kamerabasert navigasjon. Men resultatene viser også at implementering av akselerometer gjør programmet mindre robust. Dette på grunn av problemet som oppstår ved feilestimering av orienteringen. Som vil fører til at akslerasjonsmålingene som også inneholder gravitasjonen $\mathbf{g}$ blir rotert feil, og $\mathbf{g}$ blir fordelt på akser som i utgangspunktet ikke skal ha komponenter fra $\mathbf{g}$, noe som igjen fører til hastighet i gal retning. Ved implementering av gyro- og akselerasjonsmåling må det enten i tillegg også implementeres et 3-akset magnetometer, eller så må kjent informasjon om systemet brukes for å lage en syntetisk magnetometermåling. Kjent informasjon i eksperimentet i denne rapporten ville ha vært at kamerat bare kan bevege seg rundt på planet. Med denne informasjonen ville vinkelhastigheten blitt begrenset til z-aksen. Det vurderes som høyst nødvendig å implementere magnetometermålinger ved implementasjon av akselerometermålinger.

8 KONKLUSJON

Denne rapporten bruker et eksisterende kamerabasert navigasjonsprogram som benytter optiske bilder fra et ordinært monokamera for å løse SLAM problemet i 6 DOF. Rapporten presenterer en utvidelse av programmet som implementerer målinger av lineærakselerasjon og vinkelhastighet. Utvidelsen ble utført ved å bruke målingene som pådrag i tidsoppdateringen. Det har blitt utført eksperimenter ved bruk av det kamerabasert navigasjonsprogram med og uten utvidelsen.

Resultantene fra eksperimentene uten utvidelsen viser at estimatet av posisjon og orientering er uforutsigbart før antall bildemerker blir satt til ≈ 100. Estimatet blir da estimert relativt godt, men med feil skala.

Resultantene fra eksperimentene med utvidelsen viser at posisjon og orientering blir estimert godt ved kun å følge 15 bildemerker. Dermed kan antall bildemerker reduseres kraftig selv med målinger fra en lavkost treghetsplattform. Dette fører til at algoritmen reduserer behovet av datakraft. Resultatene viser også at implementering av gyro- og akselerasjonsmålinger fører til utforinger ved feilestimering av orienteringen. Rapporten foreslår en løsning av problemet ved implementasjon av et 3-akset magnetometer eller lage en syntetisk magnetometermåling ved å bruke av kjent informasjon om systemet. For at estimering av posisjon og orientering skal bli robust må magnetometermåling implementeres.

Kameranavigasjon kan brukes istedenfor eller supplere GPS navigasjon. Men en utfordring er at programmet er resurskrevende og krever derfor en kraftig datamaskin for å estimere et godt estimat ved bruk av ren kamerabasert navigasjon i sanntid. Denne rapporten viser at det derfor er aktuelt å implementere akselerasjon- og gyromålinger for å redusere antall bildemerker og dermed redusere nødvendig datakraft algoritmen trenger.

9 VIDERE ARBEID

Underkapitlene inneholder forslag til videre arbeid.

9.1 Implementere 3-akset magnetometermålinger

For å forbedre problemene orienteringsfeilen fører til er det nødvendig å utvide programmet med målinger fra et 3-akset magnetometer. Eller utnytte kjent informasjon om systemet og lage en syntetisk magnetometer måling.

9.2 Gyroskop og akselerometer med høyere frekvens

Gyro- og akselerometermålingene blir kjørt med 30Hz, samme frekvens som bildene. Ved å øke frekvensen på gyro- og akselerometermålingene som blir brukt i tidsoppdateringen vil det føre til en midling av målingene og vil dermed kunne gi en bedre prediksjon.

9.3 Finne årsaken til feil skala i estimatet ved ren kamerabasert navigasjon

Resultatene fra eksperimentet ved ren kamerabasert navigasjonen viser at posisjonen blir estimert med en feil skala. Programmet skal være i stand til å estimere posisjonen i tilnærmet riktig skala.

9.4 Ny kamera- og treghetsplattformrigg

Ved å lage en mer stabil rigg ville måleresultatene fra gyroskopet og akselerometeret innehold mindre støy. Dette vil føre til at man kan ”stole” mer på målingene og dermed kunne gi et bedre navigasjonsresultat.

9.5 Sammenligne resultater fra et nøyaktig gyroskop og akselerometer med et lavkost gyroskop og akselerometer

Det hadde vært interessant å sammenligne resultatene ved kameranavigasjon med gyro- og akselerasjonsmålingen fra et nøyaktig gyroskop og akselerometer med et lavkost gyroskop og akselerometer. FFI har gyroskop og akselerometer med forskjellig nøyaktigheter. Det kunne derfor være aktuelt å låne utstyr fra FFI for å sammenligne resultatene på lavkost og nøyaktig gyroskop og akselerometer.

9.6 Bruke filter på bildene for lettere å finne gode bildemerker

Ved å legge på ulike filter på bildene som f.eks. kontrast, hadde det vært interessant å se om dette vil føre til at mer pålitelig bildemerker blir valgt og dermed gi et bedre navigasjonsresultat.

REFERANSER

[1] J. Civera, Ó. G. Grasa, A. J. Davison, and J. M. M. Montiel, “1-Point RANSAC for EKF Filtering: Application to Real-Time Structure from Motion and Visual Odometry,” *Journal of Field Robotics*, pp. vol. 27(5), 2010.

[2] J. Civera. "1-Point RANSAC Inverse Depth EKF Monocular SLAM - Matlab Code Version 1.01," 01.02.2015, 2015; http://webdiis.unizar.es/~jcivera/code/1p-ransac-ekf-monoslam.html.

[3] "Pinhole camera model," 04.02.2015; http://en.wikipedia.org/wiki/Pinhole_camera_model.

[4] "Pinhole Camera," 15.02.2015; http://en.wikipedia.org/wiki/Pinhole_camera.

[5] "Brennvidde," Februar 5, 2015; https://snl.no/brennvidde.

[6] M. A. Fischler, and R. C. Bolles, “Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography,” *Commun. ACM*, vol. 24, no. 6, pp. 381-395, 1981.

[7] Wikipedia. "RANSAC," 18.05, 2015; http://en.wikipedia.org/wiki/RANSAC.

[8] J. L. Blanco. "RANSAC C++ examples," 18.05.2015, 2015; http://www.mrpt.org/tutorials/programming/maths-and-geometry/ransac-c-examples/.

[9] J. Civera, A. J. Davison, and J. Montiel, “Inverse Depth Parametrization for Monocular SLAM,” *Robotics, IEEE Transactions on*, vol. 24, no. 5, pp. 932-945, 2008.

[10] C. INC, “Camara User Guide Ixus 220 HS,” 2011.

[11] J.-Y. Bouguet. "Camera Calibration Toolbox for Matlab," 02.03.2015, 2015; http://www.vision.caltech.edu/bouguetj/calib_doc/.

VEDLEGG

Vedlegg A 1-Point RANSAC UKF

Algorithm 1 1-Point RANSAC EKF

INPUT: $\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{P}_{k-1|k-1}$ {EKF estimate at step $k-1$}
  th {Threshold for low-innovation points. In this paper, $th = 2\sigma\_pixels$}
OUTPUT: $\hat{\mathbf{x}}_{k|k}, \mathbf{P}_{k|k}$ {EKF estimate at step k}

{A. EKF prediction and individually compatible matches}
$[\hat{\mathbf{x}}_{k|k-1}, \mathbf{P}_{k|k-1}] = EKF\_prediction(\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{P}_{k-1|k-1}, \mathbf{u})$
$[\hat{\mathbf{h}}_{k|k-1}, \mathbf{S}_{k|k-1}] = measurement\_prediction(\hat{\mathbf{x}}_{k|k-1}, \mathbf{P}_{k|k-1})$
$\mathbf{z}^{IC} = search\_IC\_matches(\hat{\mathbf{h}}_{k|k-1}, \mathbf{S}_{k|k-1})$

{B. 1-Point hypotheses generation and evaluation}
$\mathbf{z}^{li\_inliers} = [\,]$
$n_{hyp} = 1000$ {Initial value, will be updated in the loop}
for $i = 0$ to n_{hyp} **do**
  $\mathbf{z}_i = select\_random\_match(\mathbf{z}^{IC})$
  $\hat{\mathbf{x}}_i = EKF\_state\_update(\mathbf{z}_i, \hat{\mathbf{x}}_{k|k-1})$ {Notice: only state update; NO covariance update}
  $\hat{\mathbf{h}}_i = predict\_all\_measurements(\hat{\mathbf{x}}_i)$
  $\mathbf{z}_i^{th} = find\_matches\_below\_a\_threshold(\mathbf{z}^{IC}, \hat{\mathbf{h}}_i, th)$
  if $size(\mathbf{z}_i^{th}) > size(\mathbf{z}^{li\_inliers})$ **then**
    $\mathbf{z}^{li\_inliers} = \mathbf{z}_i^{th}$
    $\epsilon = 1 - \frac{size(\mathbf{z}^{li\_inliers})}{size(\mathbf{z}^{IC})}$
    $n_{hyp} = \frac{log(1-p)}{log(1-(1-\epsilon))}$
  end if
end for

{C. Partial EKF update using low-innovation inliers}
$[\hat{\mathbf{x}}_{k|k}, \mathbf{P}_{k|k}] = EKF\_update(\mathbf{z}^{li\_inliers}, \hat{\mathbf{x}}_{k|k-1}, \mathbf{P}_{k|k-1})$

{D. Partial EKF update using high-innovation inliers}
$\mathbf{z}^{hi\_inliers} = [\,]$
for every match $\mathbf{z}^j$ above a threshold th **do**
  $[\hat{\mathbf{h}}^j, \mathbf{S}^j] = point\_j\_prediction\_and\_covariance(\hat{\mathbf{x}}_{k|k}, \mathbf{P}_{k|k}, j)$
  $\nu^j = \mathbf{z}^j - \hat{\mathbf{h}}^j$
  if $\nu^{j\top} \mathbf{S}^{j-1} \nu^j < \chi^2_{2,0.01}$ **then**
    $\mathbf{z}^{hi\_inliers} = add\_match\_j\_to\_inliers(\mathbf{z}^{hi\_inliers}, \mathbf{z}^j)$ {If individually compatible, add to inliers}
  end if
end for
if $size(\mathbf{z}^{hi\_inliers}) > 0$ **then**
  $[\hat{\mathbf{x}}_{k|k}, \mathbf{P}_{k|k}] = EKF\_update(\mathbf{z}^{hi\_inliers}, \hat{\mathbf{x}}_{k|k}, \mathbf{P}_{k|k})$
end if

Vedlegg B MATLAB kode - Utvidelse av kameramodellen

```
function X_k_km1=fv(X_k_k,delta_t, type, std_a, std_alpha,
acc_data, gyro_data)

rW =X_k_k(1:3,1);
qWR=X_k_k(4:7,1);
vW =X_k_k(8:10,1);
wW =X_k_k(11:13,1);

% Acc
g_vector = [0,-9.80665,0]';
mean_acc = [0.2347 0.0789 0.30619];
acc_data_w_g = q2r(qWR)*(acc_data-mean_acc)';
acc_data_w = acc_data_w_g - g_vector;
acc_data_w_velocity  = acc_data_w*delta_t;
```

```
% Gyro
mean_gyro = [-0.0090 -0.0116 0.0233];
gyro_data_w = q2r(qWR)*(gyro_data-mean_gyro)';


if strcmp(type,'constant_velocity')
    X_k_km1=[rW + (vW + acc_data_w_velocity)*delta_t;
        reshape(qprod(qWR,v2q(gyro_data_w*delta_t)),4,1);
        vW + acc_data_w_velocity;
        gyro_data_w];
end
```

Vedlegg C MATLAB kode - Hovedrutine

```
%-------------------------------------------------------------------
-------
% 1-point RANSAC EKF SLAM from a monocular sequence
%-------------------------------------------------------------------
-------

% Copyright (C) 2010 Javier Civera and J. M. M. Montiel
% Universidad de Zaragoza, Zaragoza, Spain.

% This program is free software: you can redistribute it and/or
modify
% it under the terms of the GNU General Public License as
published by
% the Free Software Foundation. Read
http://www.gnu.org/copyleft/gpl.html for details

% If you use this code for academic work, please reference:
%   Javier Civera, Oscar G. Grasa, Andrew J. Davison, J. M. M.
Montiel,
%   1-Point RANSAC for EKF Filtering: Application to Real-Time
Structure from Motion and Visual Odometry,
%   to appear in Journal of Field Robotics, October 2010.

%-------------------------------------------------------------------
-------
% Authors:  Javier Civera -- jcivera@unizar.es
%           J. M. M. Montiel -- josemari@unizar.es

% Robotics, Perception and Real Time Group
% Aragón Institute of Engineering Research (I3A)
% Universidad de Zaragoza, 50018, Zaragoza, Spain
% Date   :  May 2010
%-------------------------------------------------------------------
-------

clear variables; close all; clc;
rand('state',0); % rand('state',sum(100*clock));

%-------------------------------------------------------------------
-------
% Sequence, camera and filter tuning parameters, variable
initialization
%-------------------------------------------------------------------
-------
```

```
% Read acc and gyro data
read_acc_gyro;

% Camera calibration
cam = initialize_cam;

% Set plot windows
set_plots;

% Sequence path and initial image
sequencePath =
'../sequences/NOTE_II_ACC_GYRO_FILM4_320/rawoutput';
initIm = 1;
lastIm = 3040;

% Initialize state vector and covariance
[x_k_k, p_k_k] = initialize_x_and_p;

% Initialize EKF filter
sigma_a = 2; % standar deviation for linear acceleration noise
sigma_alpha = 0.75; % standar deviation for angular acceleration
noise
sigma_image_noise = 1.0; % standar deviation for measurement
noise

filter = ekf_filter( x_k_k, p_k_k, sigma_a, sigma_alpha,
sigma_image_noise, 'constant_velocity' );

% variables initialization
features_info = [];
trajectory = zeros( 7, lastIm - initIm );
% other
min_number_of_features_in_image = 75;
generate_random_6D_sphere;
measurements = []; predicted_measurements = [];

%--------------------------------------------------------------------
% Main loop
%--------------------------------------------------------------------
writerObj = VideoWrit-
er('NOTE_II_ACC_GYRO_FILM4_320_gyro&acc_sigma_a2_sigma_alpha0.75_
treshold20_maxatt150_features75.avi');
writerObj.FrameRate = 30;
open(writerObj);

im = takeImage( sequencePath, initIm );

for step=initIm+1:lastIm

    % Map management (adding and deleting features; and convert-
ing inverse depth to Euclidean)
    [ filter, features_info ] = map_management( filter, fea-
tures_info, cam, im, min_number_of_features_in_image, step );

    % EKF prediction (state and measurement prediction)
    [ filter, features_info ] = ekf_prediction( filter, fea-
tures_info, acc_data(step-1,2:4), gyro_data(step-1,2:4));
```

```
    % Grab image
    im = takeImage( sequencePath, step );

    % Search for individually compatible matches
    features_info = search_IC_matches( filter, features_info,
cam, im );

    % 1-Point RANSAC hypothesis and selection of low-innovation
inliers
    features_info = ransac_hypotheses( filter, features_info, cam
);

    % Partial update using low-innovation inliers
    filter = ekf_update_li_inliers( filter, features_info );

    % "Rescue" high-innovation inliers
    features_info = rescue_hi_inliers( filter, features_info, cam
);

    % Partial update using high-innovation inliers
    filter = ekf_update_hi_inliers( filter, features_info );

    % Plots,
    plots
    display( step );

    % Grab a frame
    frame = getframe(figure_all);
    writeVideo(writerObj,frame);

end
close(writerObj);
```

Vedlegg D MATLAB kode - Initialisere kameraparameter

```
function cam = initialize_cam()
% IXUS 220 HS 06.05.2015
% Sensor size
sx = 6.16;
% sy = 4.62;

% Bildeoppløsning
nCols = 320;
nRows = 240;

% Pixel størrelse
d =     sx/nCols;
% Forvrengning i bilde
k1 = 0.00105241745685872;
k2 = 3.03988951947260e-05;
% Fokallengde i mm
f = 4.31652544678820;
% % Bildesenter oppgitt i antall pixler
Cx = 1.594212639547530e+02; %Test av nytt senter
Cy = 1.291688197874897e+02;

cam.k1 =    k1;
```

```
cam.k2 =    k2;
cam.nRows = nRows;
cam.nCols = nCols;
cam.Cx =    Cx;
cam.Cy =    Cy;
cam.f =     f;
cam.dx =    d;
cam.dy =    d;
cam.model = 'two_distortion_parameters';

cam.K =     sparse( [ f/d   0     Cx;
                0  f/d    Cy;
                0    0     1] );
```